

SRS PRISMA GUIDE

Version 4.1.1

Copyright (c) 2005 LION bioscience AG (LION). All rights reserved.

LION bioscience SRS 8.1.1 Documentation

This manual, as well as the software described in it, is furnished under license and may only be used or copied in accordance with the terms of such license. The information in this manual is furnished for information only, is subject to change without notice, and should not be construed as a commitment by LION bioscience AG. LION bioscience AG assumes no responsibility or liability for any errors or inaccuracies that may appear in this book.

Except as permitted by such license, no part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior permission of LION bioscience AG.

c-Tree is a (registered) trademark of the Faircom Corporation.

There may be visual deviations between graphics in the manuals and the released software.

Comments about the documentation are welcome at: documentation@uk.lionbioscience.com

For customer support issues please e-mail support@uk.lionbioscience.com

SRS PRISMA GUIDE

CHAPTER 1: INTRODUCING SRS PRISMA.....	1
1.1 What is SRS Prisma?	2
1.2 How Does SRS Prisma Work?	3
1.2.1 The SRS Prisma Staging Strategy	3
1.2.2 The Prisma Staging Process	3
1.2.3 Anatomy of a SRS Prisma Run	5
1.2.3.1 Step 1: Analysis	5
1.2.3.2 Step 2: Check and Update	6
1.2.3.3 Step 4: Check and Link Completed Libraries	7
1.2.3.4 Step 5: Block Failed Libraries and move Complete Libraries	7
1.2.3.5 Step 5: Check for 'Online' Libraries	8
1.2.3.6 Step 6: Quality Inspection	8
1.2.3.7 Step 7: Generate Report	8
1.2.3.8 Step 8: Archive Reports	8
1.3 What's New for SRS Prisma 4	9
1.3.1 Prisma Configuration	9
1.3.2 New Remote Update Module	9
1.3.3 Support for Remote Updating of Indices	10
1.3.4 Support for External Trigger Mechanisms	10
1.3.5 Support for Pre-indexing Commands	10
1.3.6 Fail-over Command Support for Unpack and Reformat Commands	10
1.3.7 Easy Configuration of Production of FASTA/BLAST Files	10
1.3.8 New Scheduling Mechanism	11
1.3.9 New Installation Controls	11
1.3.10 Optimized Staging Process	11
1.3.11 Resource Usage Reports	11
1.3.12 Mail-based Reporting	12

1.3.13	Extended Quality Reports	12
1.4	What's New in SRS Prisma 4.1	12
1.4.1	Automatic configuration file updating	12
1.4.2	Per-library control of batch queue systems	12
1.4.3	Forced reformatting of libraries	13
1.4.4	Improved robustness of blocking behavior	13
1.4.5	Improved clarity of decision reports	13
1.4.6	Extension of local copy phase to include indices	13
1.4.7	'Smart' runPrisma locking	14
1.4.8	Modularization of Quality Report Module	14
1.5	What's New in SRS 8.1.1	14
1.5.1	Exclusion of files from remote checking	14
CHAPTER 2:	INSTALLING SRS PRISMA	15
2.1	Requirements for SRS Prisma	16
2.2	Installing SRS Prisma	16
2.3	Migrating to SRS Prisma 4.1	19
2.3.1	Importing a Global Configuration File	19
2.3.2	Converting Resource Objects	19
CHAPTER 3:	CONFIGURING SRS PRISMA	21
3.1	Introduction	22
3.2	Global Configuration	22
3.2.1	How to Alter Global Configuration Settings	22
3.2.2	Remote Configuration	26
3.2.2.1	General Settings	27
3.2.2.2	Using an FTP Proxy	28
3.2.2.3	Using an HTTP Proxy	31

3.2.2.4 Testing a Remote Connection	32
3.3 Execution Configuration	33
3.3.1 General Settings	33
3.3.1.1 Always honor PRISMA_RUNNING lockfile	33
3.3.1.2 Max. Repetitions of Failed Updates	34
3.3.1.3 Make Command	34
3.3.1.4 Maximum Parallel Processes	34
3.3.1.5 Shell Command	34
3.3.1.6 Suppress Warnings from srsbuild	35
3.3.1.7 Parser Logging Level	35
3.3.2 Using a Batch Queue System	35
3.3.2.1 Platform LSF	36
3.3.2.2 Sun Grid Engine	37
3.3.2.3 Other Systems	37
3.3.2.4 Configuration of Batch Queue System	38
3.3.2.5 Batch Queuing and Quote Escaping	41
3.3.2.6 Testing a Batch Queue System	41
3.4 Check Configuration	41
3.4.1 Show Verbose Output	43
3.4.2 Show Debugging Output	43
3.4.3 Carrying Out Local Check Only	43
3.4.4 Carry Out Check of Online Files Only	43
3.4.5 Check All Links	43
3.4.6 Do Not Check links	43
3.4.7 Use COMPLETED Flag Checking	44
3.4.8 Preserve Offline Symbolic Links	44
3.4.9 File Timestamp to Use	44
3.4.10 Libraries to Check	45
3.4.11 Libraries to Exclude from Checking	45

3.4.12 Groups to Check	45
3.4.13 Groups to Exclude from Checking	45
3.5 Installation Configuration	45
3.5.1 Pre-installation Checks	45
3.5.2 Installation Options	47
3.5.3 Command to Run Before Installation	47
3.5.4 Command to Run After Installation	47
3.5.5 Do Not Delete Uncompressed files	47
3.5.6 Move Libraries Online Regardless of Dependant Failure	47
3.6 Report Configuration	48
3.6.1 HTML Report Generation	48
3.6.2 Location for HTML Reports	49
3.6.3 Generate Dependency Tree Images	49
3.6.4 Number of past runs in usage report	49
3.6.5 Generate Quality Reports	49
3.6.6 Generate Trace Reports	50
3.6.7 Mail-based Reporting	50
3.6.8 Quality Reports	50
3.7 Configuration update	51
3.8 Configuring a Library for Use with SRS Prisma	52
3.8.1 Configuring a Library for Remote Update	56
3.8.1.1 Remote Data Updates	56
3.8.1.2 Local Locations	56
3.8.1.3 Specifying Remote Locations	56
3.8.1.4 FTP	58
3.8.1.5 HTTP	61
3.8.1.6 HTTPS	62
3.8.1.7 File	63

3.8.1.8 SCP	64
3.8.1.9 General Options	66
3.8.1.10 Specifying Remote Files	67
3.8.1.11 Handling Archives	70
3.8.1.12 Alternative File Comparison Techniques	71
3.8.1.13 Specifying Preprocessing Commands	72
3.8.1.14 Advanced Options	76
3.8.1.15 Using Recursion	77
3.8.1.16 Symbolic Links	77
3.8.1.17 Sanity Checking	77
3.8.1.18 Remote File Size Comparison	78
3.8.1.19 Maximum Parallel Downloads	78
3.8.1.20 Local Contents File	79
3.8.1.21 Testing Remote Settings	79
3.8.2 Remote Index Updates	79
3.8.2.1 Specifying Remote Hosts	80
3.8.2.2 Specifying Remote Index Files	82
3.8.2.3 Specifying Preprocessing Commands	84
3.9 Configuring the Indexing Phase	85
3.9.1 Indexing Settings	86
3.9.2 Non-SRS Libraries	87
3.9.3 Pre-indexing Commands	87
3.10 Configuring Postprocessing	88
3.10.1 Generating FASTA and BLAST Files from Sequence Data	89
3.10.1.1 Query-Based FASTA Generation	90
3.10.1.2 Files-Based FASTA Generation	92
3.10.1.3 Use of Existing FASTA Files	95
3.10.1.4 BLAST File Generation	96
3.10.2 Running Reformat Commands	98
3.10.2.1 Postprocessing Failover Commands	100

3.10.2.2 Postprocessing and Compressed Files	101
3.11 Installation Settings	102
3.11.1 Installation Mechanisms	102
3.11.1.1 Move New Files Online (move)	103
3.11.1.2 Switch Symbolic Links (switchlink)	104
3.11.1.3 Use Specified Commands for Installation (command)	104
3.11.1.4 Do Not Carry Out Installation (none)	105
3.11.2 Archiving	106
3.11.2.1 Do Not Archive Old Data and Index Files (delete)	106
3.11.2.2 Archive by Adding Extension to Filename (rename)	106
3.11.2.3 Archive by Moving to Another Location (move)	107
3.11.2.4 Archiving and Online-Offline Data Directories	107
3.11.3 Installation Commands	107
3.11.4 Installation Checks	109
3.11.4.1 Hierarchy Checking	109
3.11.4.2 Specified Checks	109
3.11.4.3 Time-based Checking	110
3.11.4.4 Command-based Checking	111
3.11.4.5 Query-based Checking	112
3.12 Schedule Settings	113
3.12.1 Scheduling for Automatic Updates	113
3.12.2 Schedule Frequency	115
3.12.3 Schedule Control	116
3.12.4 Advanced Scheduling Control	117
3.13 Library-specific batch queue commands	118
3.14 Dependent Libraries	119
3.14.1 Configuring a Library as Dependent	119
3.14.2 Configuring a Library as Dependent on Online Data	121

3.14.3 Specifying Preprocessing Commands	121
3.14.4 Non-SRS Dependent Libraries	123
3.14.5 Uses of Dependent Libraries	123
3.14.6 Limitations of Dependent Libraries	124
3.14.7 Building Dependent Libraries Manually	124
3.14.8 Dependent Library File Lists	124
3.14.8.1 Using the Library.files Attribute	125
3.14.8.2 Inheriting File Information from Parents	126
3.15 External Update Triggers	128
CHAPTER 4: RUNNING SRS PRISMA	133
4.1 Introduction	134
4.2 Running SRS Prisma Manually	134
4.2.1 Option Flags	135
4.3 Running SRS Prisma Automatically	139
4.4 During a SRS Prisma Run	140
4.4.1 Stopping SRS Prisma	141
4.4.2 Restarting SRS Prisma	141
4.5 Moving Libraries Manually	142
4.6 Advanced Use of SRS Prisma	144
4.6.1 Split Runs	144
4.6.2 Run State Preservation	145
CHAPTER 5: SRS PRISMA UPDATE REPORTS	147
5.1 Introduction	148
5.2 Creating Reports	148
5.2.1 Types of Report	148

5.2.2 Command Line Arguments	149
5.2.3 Methods for Producing Reports	150
5.3 Viewing Reports	150
5.3.1 SRS Prisma Status Color Key	150
5.3.2 Getting Help	151
5.3.3 The Calendar Page	151
5.3.4 Update Report Page	153
5.3.4.1 Examples of Updated Library Graphics	156
5.3.4.2 Examples of Updated Link Graphics	157
5.3.5 Stage Reports	158
5.3.5.1 General Format of Reports	158
5.3.5.2 The Local Copy Report	159
5.3.5.3 The Download Report	159
5.3.6 The Pretranslation Report	161
5.3.7 The Build Report	163
5.3.8 The Reformat Report	167
5.3.9 The Link Stage Report	167
5.3.10 The Link Report	168
5.3.11 The Move Report	172
5.3.12 The Target Report	173
5.3.13 The Dependency Tree	176
5.3.14 The Decision Report	178
5.3.15 The Usage Report	184
5.3.16 The Quality Report	187
CHAPTER 6: SRS PRISMA - TROUBLESHOOTING	189
6.1 Introduction	190
6.2 Finding Errors	190

6.3 Isolating Errors	191
6.4 Manually Tracking Datafiles	191
6.5 Correcting Errors	192
CHAPTER 7: SRS PRISMA QUALITY REPORT	193
7.1 What is SRS Prisma Quality Report?	194
7.2 Tests in Prisma Quality Report	194
7.3 Running SRS Prisma Quality Report	195
7.3.1 Running SRS Prisma Quality Report with Prisma	195
7.3.2 Running SRS Prisma Quality Report Stand-Alone	195
7.3.3 Running SRS Prisma Quality Report from the Commandline	196
7.4 Error Checking Activities for SRS Relational	198
7.5 Error Checking Activities for Integrated Tools in SRS	199
7.5.1 Launching the Tools Test	199
7.6 Error Types Used in SRS Prisma Quality Report	200
7.7 Quality Report HTML Pages	201
7.7.1 Main Page	202
7.7.2 Summary Report	202
7.7.3 Databank Group Reports	204
7.7.3.1 Left-Hand Chart	205
7.7.4 Right-Hand Chart	206
7.8 Tools Test Results Page	206
7.9 Error Details Page	207
7.9.1 Error Summary Report	208
7.9.2 Quick References Toolbox	209
7.9.3 Full Error Details Classified by Type Reports	209

7.9.4 Monthly Errors Breakdown	210
7.9.5 Data-Fields Index Status	211
7.10 Archiving	212
7.11 Tests Used	212
CHAPTER 8: CONFIGURATION UPDATES WITH SRS PRISMA.....	217
8.1 Introduction	218
8.1.1 The configuration file update mechanism	218
8.2 Using the update mechanism	221
8.2.1 Running as standalone	222
8.2.2 Running as part of Prisma	224
8.2.3 Conflict Resolution	225
8.3 Update Reports	229
8.4 Configuration of the update mechanism	233
8.4.1 Basic configuration	235
8.4.2 Editing remote configurations	235
8.4.3 Editing remote mirror locations	239
8.4.4 Editing remote hosts	240
8.4.5 Editing remote files	241
8.4.6 Editing memorized decisions	241
8.5 Trouble-shooting	242
CHAPTER 9: SRS PRISMA - WORKED EXAMPLES	245
9.1 Introduction	246
9.2 Single File Libraries	246
9.2.1 Simple Updating	246
9.2.2 Unpacking Flat-Files	247

9.2.3 Reformatting Flat-Files	249
9.3 Handling Multiple Files	251
9.4 Dependent Libraries	252
9.4.1 A Simple Dependent Library	252
9.4.2 Dependent Libraries and File Information	253
9.4.3 Non-SRS Libraries for Data Manipulation	257
9.5 Dependent Non-SRS Libraries	259
CHAPTER 10: SRS PRISMA - COMMON ERRORS	261
10.1 Introduction	262
10.2 Problem: Changes to a Resource Object Are Not Recognized	262
10.3 Problem: SRS Prisma Tries to Move a Non-Existent File Online and Fails	263
10.4 Problem: The Downloading Fails Frequently because the FTP Connection is Unreliable	263
10.5 Problem: SRS Prisma Reports that Files Need to be Updated, but Downloading always Fails	263
10.6 Problem: The Unpack Command Contains a Mangled Filename, e.g. gunzip uniprot_sprot.dat.gz.gz	263
10.7 Problem: SRS Prisma Tries to Run srsbuild Commands with -parts 0 on a Dependent Library	264
10.8 Problem: SRS Prisma Keeps Trying to Rebuild the Library Even Though No New Files Are Downloaded	264
10.9 Problem: All the Files for the Library Have Been Deleted by SRS Prisma	264
10.10 Problem: SRS Prisma Reports that No Suitable Files for Indexing Are Found	264
10.11 Problem: Remote Checking Fails for Some Libraries	265
APPENDIX A: WGETPRISMA	267

XII

CHAPTER

1

INTRODUCING SRS PRISMA



Note: If you are an experienced SRS Prisma user, you may skip to Section 1.4, What's New in SRS Prisma 4.1, page 12.

1.1 What is SRS Prisma?

Ensuring that an SRS installation is providing consistent, up-to-the-minute data to users can often be a time-consuming task for an SRS administrator. There are many tasks involved – to take the example of a sequence database like EMBL or GENBANK:

- Check local datafiles against remote FTP servers to see if new files have been added
- Any new files have to be downloaded, unpacked, and then new SRS indices produced (including links to other databases)
- Creating new files and indices for third-party applications like BLAST
- If other databases depend on these files, such as a non-redundant database, it may also be necessary to update them as well

SRS Prisma is a tool that automates the complex checking process for an SRS installation. The aim of running SRS Prisma is to get an SRS installation up-to-date as quickly as possible with minimal disruption to users. SRS Prisma can be scheduled as an automatic process, which checks and updates databases whenever necessary, and makes full use of the hardware available to the installation. All new data and indices are created in an 'offline' directory, and moved rapidly online when complete, which minimizes the length of time a database is unavailable to a user.

SRS Prisma is not limited to downloading and processing data for the express purpose of creating SRS indices. SRS Prisma can also be configured to run external commands on new and existing datafiles. For instance, FASTA and BLAST format datafiles can be created from sequence datafiles, or existing SRS databases can be queried and the results used to create new data. This data can then be indexed, creating what is termed a 'dependent database'. Dependent databases can also be independent of the SRS querying mechanism, and used as a framework to do whatever data manipulation is required.

To minimize the amount of work an administrator needs to do, SRS Prisma also provides detailed reports of what update processes were carried out, with any

failures highlighted, so they can be rapidly and easily rectified. In addition, SRS Prisma also performs a thorough quality analysis on all databases on the updated system, checking their configuration and alerting the administrator to any problems that may need attention.

1.2 How Does SRS Prisma Work?

The basic mechanism used by SRS Prisma is simple in concept and similar to that used by the core SRS applications `srscheck` and `srsdo`. Internally, SRS Prisma uses an analogous Icarus application, `prismacheck`, to examine all libraries in the installation, and write any commands that need to be run to a makefile. The makefile is then executed using GNU `make`. The use of a makefile allows the complex dependencies involved in updating libraries to be easily managed, and the use of GNU `make` allows multiple processes to be run in parallel.

1.2.1 The SRS Prisma Staging Strategy

However, the way in which this mechanism is deployed for SRS Prisma is somewhat more complex. Because of the occasionally error-prone processes of downloading and indexing large datafiles, SRS Prisma has been designed to be more robust and forgiving of errors. This is manifest in the *staging* strategy used by SRS Prisma, which sequentially checks and updates groups of libraries offline, and decides what further action (such as moving, linking, and building of dependent libraries) needs to be taken on the basis of the success or failure of the update process for each library.

1.2.2 The Prisma Staging Process

To illustrate how staging works, consider the following hypothetical situation:

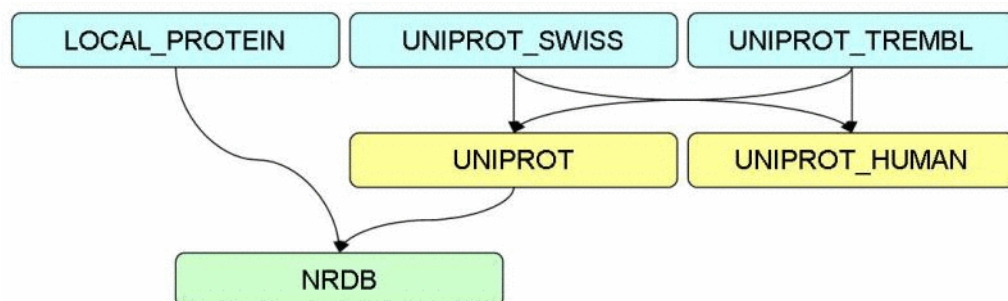


Figure 1.1 A hierarchy of libraries.

Three libraries, UNIPROT_SWISS, UNIPROT_TREMBL and LOCAL_PROTEIN (hypothetical), can be updated from an external source e.g. a remote server, or a set of local files. When either UNIPROT_SWISS or UNIPROT_TREMBL are updated, they are used to generate the virtual libraries UNIPROT and UNIPROT_HUMAN (hypothetical). UNIPROT is also part of a hypothetical dependent library called NRDB, together with the library LOCAL_PROTEIN.

The update procedure used by SRS Prisma (simplified for illustrative) would be as follows:

- Update UNIPROT_SWISS, UNIPROT_TREMBL and LOCAL_PROTEIN
- Build links involving these libraries.
- Update UNIPROT and UNIPROT_HUMAN
- Build links involving UNIPROT and UNIPROT_HUMAN.
- Update NRDB
- Build links (if any) involving NRDB
- Move all libraries online

However, each stage of the update process is monitored for failures, and this information is used to build links, virtual libraries and dependent libraries. In our example, if LOCAL_PROTEIN failed to be updated correctly, NRDB would still be updated, using the currently available online version of LOCAL_PROTEIN. Likewise, the links between UNIPROT_SWISS/UNIPROT_TREMBL and LOCAL_PROTEIN would be built using the online version of UNIPROT_PROTEIN.

1.2.3 Anatomy of a SRS Prisma Run

The following diagram illustrates how SRS Prisma implements staging:

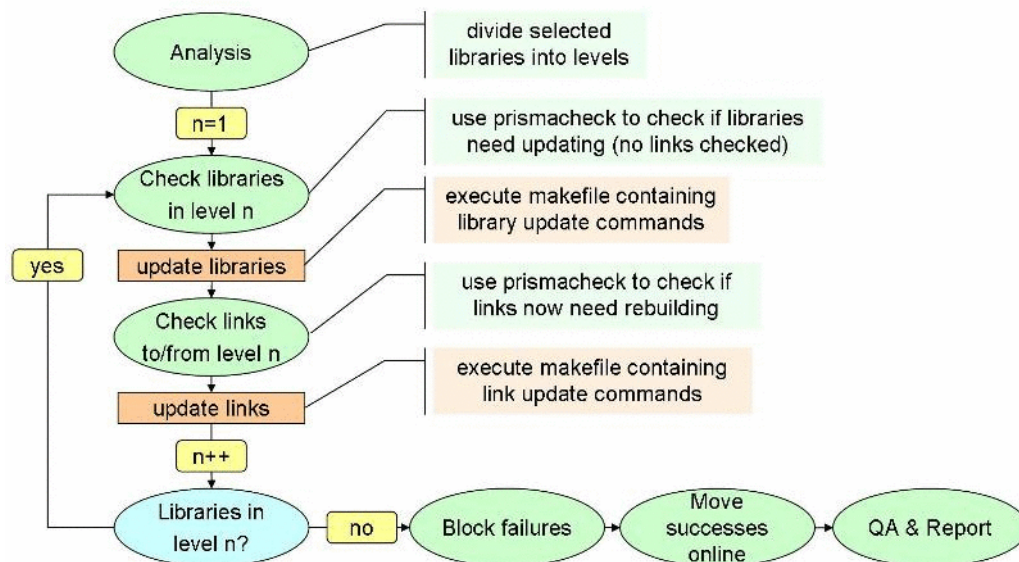


Figure 1.2 The staging update process.

The staged update process can be divided into a number of steps:

1. Analysis
2. Check and update
3. Move complete libraries
4. Check for 'online' libraries
5. Quality inspection
6. Generate Report
7. Archive Reports

1.2.3.1 Step 1: Analysis

In this step, all libraries are divided into groups, called 'levels', according to their dependency on other libraries. Level 1 libraries have no dependencies on other

libraries (e.g. are updated from a remote FTP site like UNIPROT_SWISS). Level 2 libraries depend on level 1 libraries (e.g. virtual libraries like UNIPROT). Level 3 libraries depend on Level 2 libraries (e.g. dependent libraries like UNIPROT_HUMAN) and so on and so forth.

A special type of level is for the dependent libraries of type 'createfromonline'. These libraries depend on the successful installation of other libraries and are discussed further in Section 3.14.2, Configuring a Library as Dependent on Online Data, page 121.

1.2.3.2 Step 2: Check and Update

This step is carried out once for each library level determined in step 1. First, each library in the active level is checked to see if updating is required. This process goes through the following steps until an action is found:

- *either* comparing online and offline local datafiles to those in remote repositories to find new remote files, new offline files and obsolete online files (when remote updating has been enabled)
- *or* comparing dates of offline and online files to find if offline data is newer (when remote updating has not been enabled)
- comparing dates of online data files to the ID indices to see if the online data is newer than the indices
- checking to see if all indices and associated files are present, readable and up-to-date.
- checking to see if the members of a virtual library have been updated in previous levels
- checking to see if the parents of a dependent library have been updated in previous levels

Once the update status of each library has been determined, all commands required for the update are written to a makefile, which is then executed.

The types of update commands for a library can be divided into a number of different stages:

- **LocalCopy**
Existing online datafiles and indices that do not need updating are linked or copied offline ready for reindexing.

- **Download**
New datafiles are downloaded from a remote server
- **Pretranslate**
New datafiles are preprocessed after download (for example, unpack compressed files) or other preprocessing commands are run
- **Build**
All offline datafiles are indexed and the SRS indices stored offline
- **Reformat**
Any additional post-processing is carried out after building (for example, creation of BLAST indices)

Each stage is monitored for success or failure, and this information is used to influence what happens in successive stages.

1.2.3.3 Step 4: Check and Link Completed Libraries

Depending on the success or failure of the build step, the new offline data and indices are used for creating SRS link indices with other libraries. If the library failed, any other links involving this library that need to be built are built using online data and indices (if available). In addition, online link indices between libraries that have not been updated may also need rebuilding if they are missing, corrupt or out-of-date.

1.2.3.4 Step 5: Block Failed Libraries and move Complete Libraries

Once all libraries in all levels have been updated, SRS Prisma checks to see which libraries were successfully created and moves any new indices and datafiles online.

In addition, to ensure consistency of data, SRS Prisma also checks to see if any dependent or virtual libraries have failed. If any are found, the 'member', 'peer' or 'parent' libraries are blocked from being installed, and links to or from them are rebuilt using the online versions. In our example, if UNIPROT failed to be built, UNIPROT_SWISS and UNIPROT_TREMBL would *not* be installed to maintain consistency of data between related libraries. In addition, any libraries that depend on UNIPROT_SWISS and UNIPROT_TREMBL, meaning that the "peer" virtual library UNIPROT_HUMAN would also not be installed.

Control of this behavior is discussed more in Section 3.5.6, Move Libraries Online Regardless of Dependant Failure, page 47, and Section 3.11.4.1, Hierarchy

Checking, page 109. Manually moving 'blocked' libraries online is described in Section 4.5, Moving Libraries Manually, page 142.

1.2.3.5 Step 5: Check for 'Online' Libraries

If any libraries have been set to be of type `createfromonline` (see Section 3.14.2, Configuring a Library as Dependent on Online Data, page 121), SRS Prisma checks to see if they need to be updated due to a new copy of a parent being installed. If so, the 'online' library is updated and Step 2 and 3 repeated to ensure that any libraries dependent on the 'online' libraries are also updated. Each iteration of Steps 2 to 4 is referred to as a 'round'.

1.2.3.6 Step 6: Quality Inspection

Once all new libraries are installed online, the Quality Report module of SRS Prisma is used to check each active library for configuration and data errors, and generates an HTML report with its findings. The reports produced are discussed further in Chapter 7, SRS Prisma Quality Report.

1.2.3.7 Step 7: Generate Report

The status of each update process scheduled in Steps 2 to 4 are analyzed and the administrator then generates HTML reports for inspection. The reports produced are discussed further in Chapter 5, SRS Prisma Update Reports.

1.2.3.8 Step 8: Archive Reports

The generated reports for the process are then archived for later inspection. Viewing archived reports is discussed further in Section 5.3.3, The Calendar Page, page 151.

1.3 What's New for SRS Prisma 4

1.3.1 Prisma Configuration

Global configuration is now carried out using a new Icarus class, `PrismaSettings`, an instance of which is stored in the file `$SRSETC/conf/prisma`. This has been completely restructured to be easier to configure and understand, and is described fully in Section 3.2, Global Configuration, page 22. However, it is also possible to import existing settings from an SRS Prisma 3.1 installation. This is dealt with in Section 3.2.1, How to Alter Global Configuration Settings, page 22.

The main object for controlling how SRS Prisma deals with individual libraries, the `Resource` object, has also been restructured, to improve usability and clarity, and to support the new features found in SRS Prisma 4. Many attributes have been deprecated and should not be used, while others have been added, but existing `Resource` objects can still be used by SRS Prisma, or converted permanently by SRS Prisma.

SRS VisAd has a new Prisma module designed to make configuring global and library-specific aspects of Prisma easy. Using this tool is described in more detail in Chapter 3, Configuring SRS Prisma, and examples of its use are used throughout this manual.

1.3.2 New Remote Update Module

The module responsible for remote checking and updating has been completely replaced with a new module that uses a modified version of GNU `wget` to carry out updating using HTTP(S) and FTP. The unmodified version of `wget` built from public source code will not work with Prisma. The modified binary (named `wgetPrisma` to distinguish it) is supplied as part of the SRS binary tarball, and is ready to use, but all source code is also supplied and the modifications made are described in more detail in Appendix A, page 267.



Note: Perl is no longer required by SRS Prisma.

1.3.3 Support for Remote Updating of Indices

In addition to downloading new data files from remote servers, SRS Prisma now supports downloading compatible SRS indices from remote servers (if available). Use of this feature is discussed in more detail in Section 3.8.2, Remote Index Updates, page 79.

1.3.4 Support for External Trigger Mechanisms

SRS Prisma now allows libraries to be updated in response to one or more external triggers, including presence of files, exit status of specified commands. This advanced feature is discussed in more detail in Section 3.15, External Update Triggers, page 128.

1.3.5 Support for Pre-indexing Commands

One or more commands can now be specified to be run before indexing takes place (without depending on the download of new files). Use of this feature is outlined in Section 3.9.3, Pre-indexing Commands, page 87.

1.3.6 Fail-over Command Support for Unpack and Reformat Commands

Commands can now be specified to be run if unpack or reformat commands fail, allowing 'clean-up' of said commands, as described in Section 3.10.2.1, Postprocessing Failover Commands, page 100.

1.3.7 Easy Configuration of Production of FASTA/BLAST Files

The creation of FASTA files and BLAST indices from SRS libraries is now much easier to configure, allowing FASTA files to be generated from specified lists of files, or from specified SRS queries. The configuration of this feature is described in Section 3.10.1, Generating FASTA and BLAST Files from Sequence Data, page 89.

1.3.8 New Scheduling Mechanism

SRS Prisma now has a completely new scheduling mechanism to increase the level of control an administrator has over when libraries are checked, and what level of checking should be carried out. Scheduling is described in Section 3.12, Schedule Settings, page 113.

1.3.9 New Installation Controls

The installation process, where new data and indices are installed online, now accepts a number of checks on a per-library or installation-wide level before installation can take place, and commands to be run before and after installation can also be specified. SRS Prisma also works closely with the SRS web services server to ensure that libraries are not accessible whilst installation is taking place. Installation checks are discussed further in Section 3.5.1, Pre-installation Checks, page 45 and Section 3.11.4, Installation Checks, page 109.

1.3.10 Optimized Staging Process

The execution process carried out by `runPrisma` has been altered to separate the creation of links from the updating of libraries to improve robustness, and the failure of a virtual or dependent library will also block installation of libraries in its hierarchy to ensure consistency (unless otherwise specified). `runPrisma` also now uses check-pointing to allow more robust stopping and restarting of the update process. Finally, for advanced users, the facility to separate downloading and other updating into two separate processes has also been added. More details on using `runPrisma` can be found in Chapter 4, Running SRS Prisma.

1.3.11 Resource Usage Reports

The reporting process now includes the generation of a report showing the relative size of data and indices over time for individual libraries and for the installation as a whole. Usage reports are discussed in more detail in Section 5.3.15, The Usage Report, page 184.

1.3.12 Mail-based Reporting

SRS Prisma now allows brief text reports to be sent to the administrator by email. This can include brief reports at each stage of the process, or reports alerting the administrator as soon as a failure is noticed. Configuring mail-based reporting is discussed in Section 3.6.7, Mail-based Reporting, page 50.

1.3.13 Extended Quality Reports

The Quality Report module has now been extended to include new tests on tools, loaders and XML libraries. The report has also been improved to include useful information such as 'libraries most in need of attention'. The Quality Report is discussed in Chapter 7, SRS Prisma Quality Report.

1.4 What's New in SRS Prisma 4.1

1.4.1 Automatic configuration file updating

SRS Prisma now includes functionality to automatically update SRS meta-definition files from LION's official repository. By default, Icarus files from the SRSDb directory and JSP files from the SRSTAGS directory are updated, but additional directories and repositories can be added. Where local changes have been made to distributed files (e.g. in SRSSITE), changes are automatically merged, or the administrator is alerted to the presence of conflicts that require manual intervention. The update process can be run independently, or as part of the main Prisma process, and also flags libraries for reindexing where the change requires it. Chapter 5, SRS Prisma Update Reports describes this feature in more detail.

1.4.2 Per-library control of batch queue systems

Prisma 4.0 introduced support for specifying individual batch queue commands for different stages of the update process (e.g. download, reformat etc.). Prisma 4.1

extends this functionality to allow individual batch commands to be specified for different libraries (e.g. a separate command for EMBLRELEASE reformat commands), allowing an extremely fine degree of control of the scheduling of different jobs.

1.4.3 Forced reformatting of libraries

SRS Prisma 4.1 introduces a new mechanism to allow the administrator to automatically force the execution of the reformat and BLAST/FASTA file generation commands associated with a library, without the requirement for reindexing of the data. This allows new commands to be added to a library and executed without having to reindex (e.g. addition of a species-specific BLAST file to EMBLRELEASE).

1.4.4 Improved robustness of blocking behavior

To ensure in the case of an updating failure that all libraries dependent on a failed library are prevented from being installed online, the blocking mechanism has been extended to include 'peers' of a library, as described further in Section 1.2.3.4, Step 5: Block Failed Libraries and move Complete Libraries, page 7

1.4.5 Improved clarity of decision reports

The decision reports have been extended to include information on the blocking of libraries and the consequent rebuilding of links.

1.4.6 Extension of local copy phase to include indices

Where not all indices need rebuilding for a library, the local copy phase also copies indices and associated files offline. This ensures that all subsequent offline queries can be carried out reliably.

1.4.7 'Smart' runPrisma locking

The PRISMA_RUNNING lockfile is used to indicate that an instance of runPrisma is active, and prevents subsequent updates. However, if a run is interrupted and unable to finish cleanly, this flag will also prevent new runs. This behavior can be configured to detect if the corresponding runPrisma process is still active on the current machine, and remove the lockfile if not.

1.4.8 Modularization of Quality Report Module

Tests of similar function have now been grouped into test suites. This allows the administrator to select and configure specific test suites for their Quality Report run. This offers users flexibility in deciding the collection of diagnostic tests that best suites their SRS maintenance needs.

1.5 What's New in SRS 8.1.1

1.5.1 Exclusion of files from remote checking

It is now possible to specify a regular expression to match files that should be explicitly excluded from remote checking. This allows all files apart from a subset to be easily specified. This is described in more detail in Section 3.8.1.10, Specifying Remote Files, page 67

CHAPTER

2

INSTALLING SRS PRISMA

2.1 Requirements for SRS Prisma

The following items of software are required for installing SRS Prisma 4.1:

- A working SRS 8 installation
- The corresponding SRS Prisma archive e.g. `prisma4.1.tar.gz`
- A POSIX-compliant `sh` (BASH is recommended)
- Standard UNIX utilities (including `awk`, `sed`, `find`, `ps`, `xargs`)
- (Optional) A working batch queue system (e.g. Platform LSF or Sun Grid Engine)
- (Optional) `openssl` `ssh`/`scp`
- (Optional) application to generate BLAST indices e.g. NCBI `formatdb`
- (Optional) applications to generate FASTA files from sequence data files e.g. `sp2fasta`

2.2 Installing SRS Prisma

SRS Prisma installation is run automatically using the `srinstall` script to configure the installation. Many global aspects of SRS Prisma can be configured at this stage, but it is recommended that the default settings are accepted initially.

To install SRS Prisma 4.1 in an existing installation, run the following commands:



Note: If you have unpacked the SRS Prisma tarball before installing SRS, the `srinstall` command will automatically detect this and start the SRS Prisma installation process as part of the main `srinstall` process. Follow the instructions from step 3.

1. Prepare your SRS environment and move to the SRSROOT directory:

```
% cd $SRSROOT
```
2. Unpack the SRS Prisma tar archive:

```
% gunzip -c prisma.4.1.tar.gz | tar xvf -
```
3. Use the `srinstall` script to install Prisma (note that additional output and any error logging will be written to `$SRSROOT/prismaInstall.log`):

```
% ./srinstall -prisma
```

Welcome to SRS Install

Copyright (C) 1997-2004 LION bioscience AG. All Rights Reserved.

- SRS root directory is /software/srs8

Finding OS for install... installing SRS on linux7

Do you accept the additional terms and conditions specified in the
/software/srs8/README.3rdParty file (y/n): y

The README.3rdParty file contains additional licensing for other products used by SRS. Answer 'y' to accept these terms and continue with the installation.

Welcome to SRS Prisma Install (C) 1997-2004 LION bioscience AG.

- SRS root directory is /software/srs8
Installing SRS Prisma...

Setting permissions on executable files...

Installing SRS Prisma helper libraries

Package file \${SRSLION}/packages/srsprisma.ip successfully installed.

Import configuration from existing SRS Prisma installation? [n]:

At this point, the configuration file from an Prisma 3.1 installation can be imported into the new format used for Prisma 4.1. To import the file, answer 'y' and supply the path to the original file:

Configuration file to import: /software/srs71/etc/conf/prisma

Importing file /software/srs71/etc/conf/prisma

Writing new object to /software/srs8/etc/conf/prisma



Note: If you do not wish to import a configuration file at this point, it is possible to do this at a later stage. Please consult Section 2.3.1, Importing a Global Configuration File, page 19 for more details.

SRS Prisma 4.1 stores information in different attributes of the Resource object to previous releases. Existing Resource objects can be automatically converted

Convert existing Resource objects now [n]:

As indicated, SRS Prisma 4.1 stores per-library configuration information in Resource objects that have a different format to that used in SRS Prisma 3.1. Although existing

```
Resource objects (stored in .it files) can be used, they should be converted to the
new format. To do this automatically, answer 'y' and any Resource objects that need
converting will be converted:
Converting all resources...
No resources needed converting...
```



Note: If you do not wish to convert Resource objects at this point, it is possible to do this at a later stage. Please consult Section 2.3.2, Converting Resource Objects, page 19 for more details.

Next, it is possible to configure a variety of global aspects of SRS Prisma. The details of this configuration are described in Section 3.2.1, How to Alter Global Configuration Settings, page 22. To carry out configuration now, enter 'y'. Otherwise, further configuration can be carried out at a later point as detailed in Section 3.2.1, How to Alter Global Configuration Settings, page 22.

```
Configure Prisma now? [ n ]:
```

The remainder of the installation process automatically finishes setting up the SRS Prisma installation and generating documentation:

```
Creating report directories...
Generating Reports Archive page...
Building documentation...
```

```
-----
Installation of SRS Prisma is now complete!
-----
```

To run an SRS Prisma update, either run `/path/to/srs/etc/runPrisma` from the command line or run `/path/to/srs/etc/launchPrisma` as a cron/batch job

You are now ready to configure SRS Prisma to meet your local needs (as described in Chapter 3 Configuring SRS Prisma) and to run the SRS Prisma update process (as described in Chapter 4 Running SRS Prisma).

2.3 Migrating to SRS Prisma 4.1

2.3.1 Importing a Global Configuration File

The files used for configuration by SRS Prisma have changed substantially, but old versions can still be imported into SRS Prisma. This can be done as part of the install process (see above) or separately.

To import the Prisma configuration file from your old installation, run:

```
% $SRSLION/prisma/convertPrismaConfig.i -inFile /path/to/srs71/etc/conf/prisma
```

2.3.2 Converting Resource Objects

Existing `Resource` objects (stored in `.it` files) can still be used, but it is recommended that these are converted to the new format. The following command converts old-style `Resource` objects and saves the new files to their original location:

```
% $SRSLION/prisma/convertResource.i
```

This command accepts the following options:

- I**
libraries to convert
- L**
libraries to exclude from conversion
- g**
groups to convert
- G**
groups to exclude from conversion

However, it is recommended that for LION-supported libraries, the distributed `Resource` objects are used (with local alterations as necessary).

CHAPTER

3

CONFIGURING SRS PRISMA

3.1 Introduction

SRS Prisma has been designed to work 'out of the box' with all supported SRS databases, with minimal user configuration. However, most aspects of SRS Prisma can be easily configured, allowing the SRS administrator to get the most out of their local hardware configuration, or to control the updating of local SRS databases and other related datasets.

This chapter explains how to configure various global aspects of SRS Prisma to control how the update process is run, and how to configure the way SRS Prisma updates individual databases.

3.2 Global Configuration

3.2.1 How to Alter Global Configuration Settings

SRS Prisma provides three different ways of altering global configuration settings, according to the needs and preferences of the user. This section explains how to use each of these.

The first is to use the Visual Administration Toolbox (VisAd) Prisma Module. For more information on setting up this tool, please consult Chapter 6, SRS VisAd, page 115 of the *SRS Basic Administrator's Guide*. To use the Prisma Module, start VisAd and double-click on the Prisma icon, or from the VisAd **Session** menu select **Session > Open > SRS Prisma**:

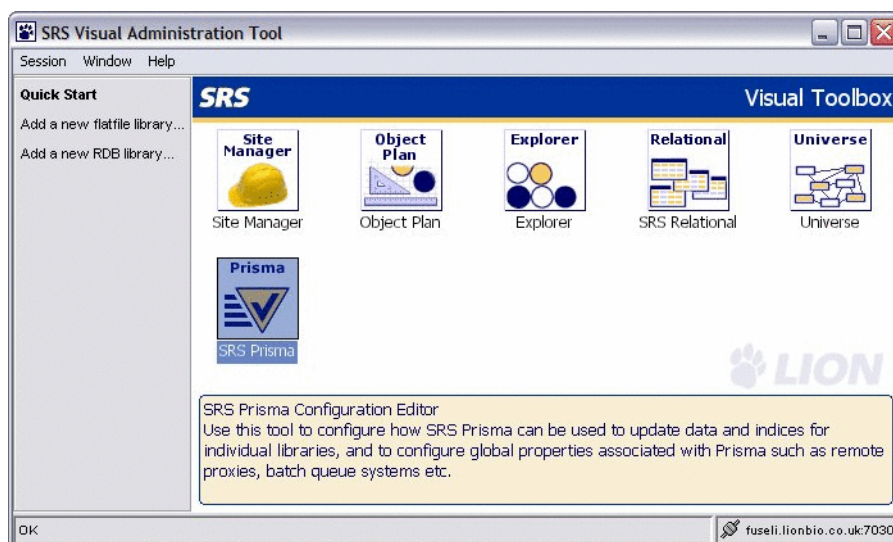


Figure 3.1 The Visual Administration toolbox (VisAd).

The main panel of the Prisma tool is divided into five tabs representing five different global aspects of SRS Prisma that can be configured. Throughout the following section, these tabs are referred to as the **Remote Configuration** tab, **Execution Configuration** tab, **Check Configuration** tab, **Installation Configuration**, tab **Report Configuration** tab and **Configuration Update** tab.

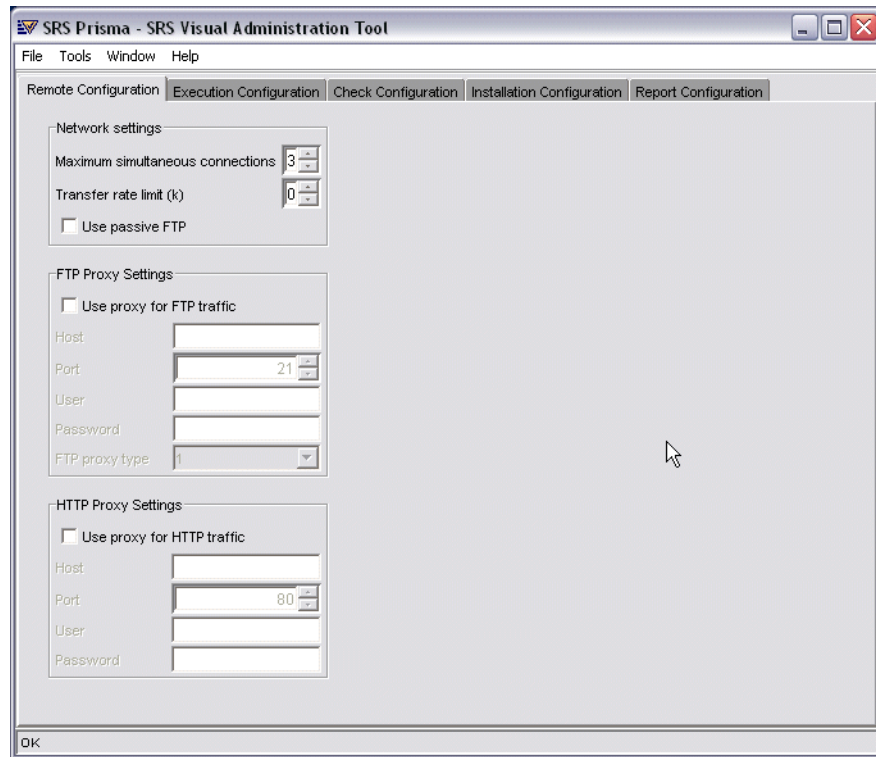


Figure 3.2 The Prisma tool main panel

To alter the desired property, select the appropriate tab, and make the change. To commit this change, open the **File** menu and select **Save configuration**.

To reject the change and to reset to the last saved set of changes, open the **File** menu and select **Revert configuration**. If the tool is closed before changes are saved, you will be asked if you wish to save the data first.

Although it is recommended that the administrator use VisAd for configuring Prisma, a command line configuration tool is also provided for use. To run this tool, execute:

```
% $SRSLION/prisma/configurePrisma.i
```

The configuration tool allows configuration by asking for answers to a series of questions, with the current values provided as default values. To make the process more convenient, these questions have been divided into the same categories as VisAd; namely **Remote Configuration**, **Execution Configuration**, **Check**

Configuration, Installation Configuration, Report Configuration and Configuration Update. For example, to set the maximum number of parallel processes, run the configure script, choose to configure the execution settings and then answer as appropriate to the question on parallel processes. At the end of the script, you can choose to save your changes or quit without saving. Note that where possible, the same terminology and questions have been used as in VisAd.

```
SRS Prisma Global Configuration
```

```
-----
```

```
Reading configuration from /srs/srs81/etc/conf/prisma
```

```
Remote Settings
```

```
-----
```

```
Configure remote settings? : [ y ] n
```

```
Execution Settings
```

```
-----
```

```
Configure settings for execution? : [ y ] y
```

```
Maximum times to repeat failed update commands : [ 1 ]
```

```
Make command to use : [ /srs/srs8cvs/bin/linux7/gmake -k ]
```

```
Maximum number of parallel update processes : [ 1 ] 8
```

```
Shell command to use : [ /bin/bash ]
```

```
Suppress warnings from srsbuild? : [ n ]
```

```
Parser logging level : [ 0 ]
```

```
Use batch queuing system? : [ n ]
```

```
Check Settings
```

```
-----
```

```
Configure settings for checking? (advanced users only) : [ n ]
```

```
Installation Settings
```

```
-----
```

```
Configure settings for installation of data? : [ y ] n
```

```
Report Settings
```

```
-----
```

```
Configure settings for production of update reports? : [ y ] n
```

```
Configuration Update Settings
```

```
-----
```

```
Configure Prisma meta-data update settings? : [ y ] n
```

```
Configuration complete!
-----
Write changes to configuration file? : [ y ] y
Writing new configuration to /srs/srs81/etc/conf/prisma
```

Finally, it is also possible (though not recommended) for experienced users who are familiar with Icarus object syntax to directly edit the global configuration file `$SRSETC/conf/prisma`. This contains an instance of the `PrismaSettings` class, which has a number of attributes containing objects representing each of the categories (`PrismaRemoteSettings`, `PrismaExecutionSettings` etc.) which in turn have attributes representing each of the global configuration aspects that users might want to alter. The details of this object are beyond the scope of this tutorial, but the appropriate classes are defined in `$SRSLION/prisma/prismaSettings.i` for those who are interested.



Note: Any changes to this file do not require `srssection` to be run, as this file is read at run-time by Prisma.

Each of the next sections deals with the configuration of each of the different aspects of Prisma. It is assumed that VisAd will be used for configuration, but the equivalent changes can always be readily made using the appropriate section of the command line tool or by editing the appropriate Icarus object directly.

3.2.2 Remote Configuration

One of the main roles of SRS Prisma is to check remote repositories for new data and index files, and download new files if required. To configure general aspects of this process, use the **Remote Configuration** tab of VisAd:

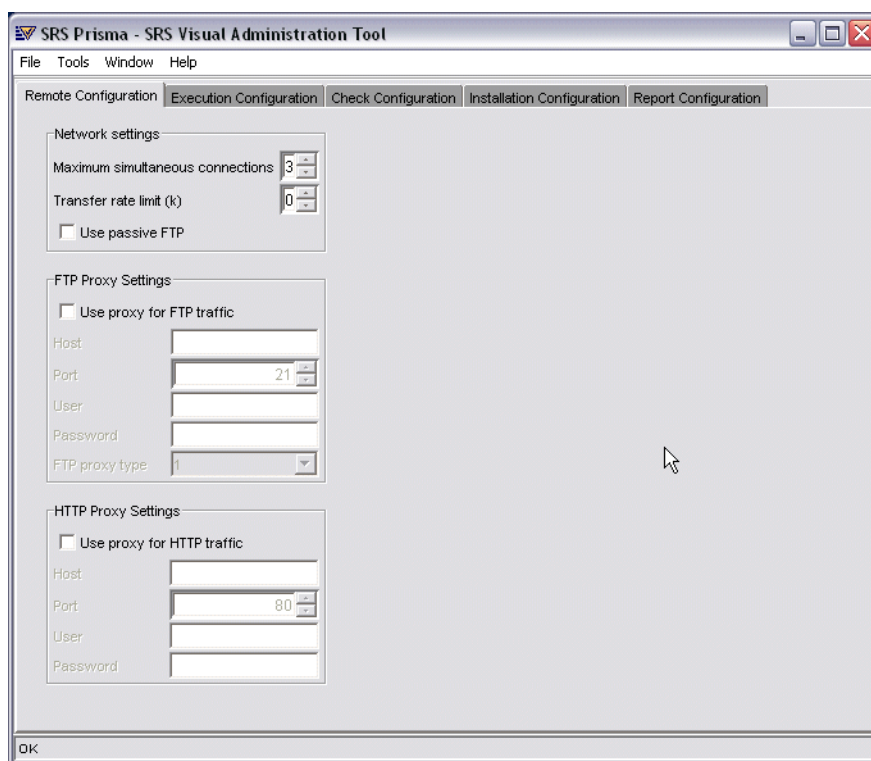


Figure 3.3 The Remote Configuration tab

The following sections describe the configuration process.

3.2.2.1 General Settings

Maximum Simultaneous Connections

SRS Prisma can be configured to restrict the maximum number of simultaneous network connections during an update. This can be used to prevent the swamping of a local or remote network with excessive connections. If this number is exceeded, new network connections will wait for a free 'slot' before being created. Each connection creates a lock file in the flags directory (`$SRSPRISMA/<run_name>/flags`), and new file transfers will not start until the number of flags falls below the value set. This means that if an individual connection fails unexpectedly, it will

continue to occupy a channel for the remainder of the operation. If all the channels become occupied in this way, the waiting connection will eventually timeout and SRS Prisma will terminate with an appropriate error status.

To free channels manually, the lock file(s):

```
$SRSPRISMA/<run_name>/dbSync.lock.HOSTNAME.nnnn
```

(where nnnn is the process ID), should be deleted.

Transfer Rate Limit

SRS Prisma can also instruct `wgetPrisma` to limit the transfer rate limit of each connection by setting the transfer rate limit to a maximum number of Kbytes per second. By default, this is set to 0 (zero), which is interpreted as unlimited bandwidth per connection.

Passive FTP

Some firewalls do not allow active FTP connections, but require the use of passive FTP. This option forces Prisma to use passive FTP.

3.2.2.2 Using an FTP Proxy

Although FTP is generally carried out directly between client and server, there are a number of products available that direct FTP traffic through a 'gateway' or 'proxy' to control access. SRS Prisma supports a wide range of such products. The proxy details can be set in the **Remote Configuration** tab.

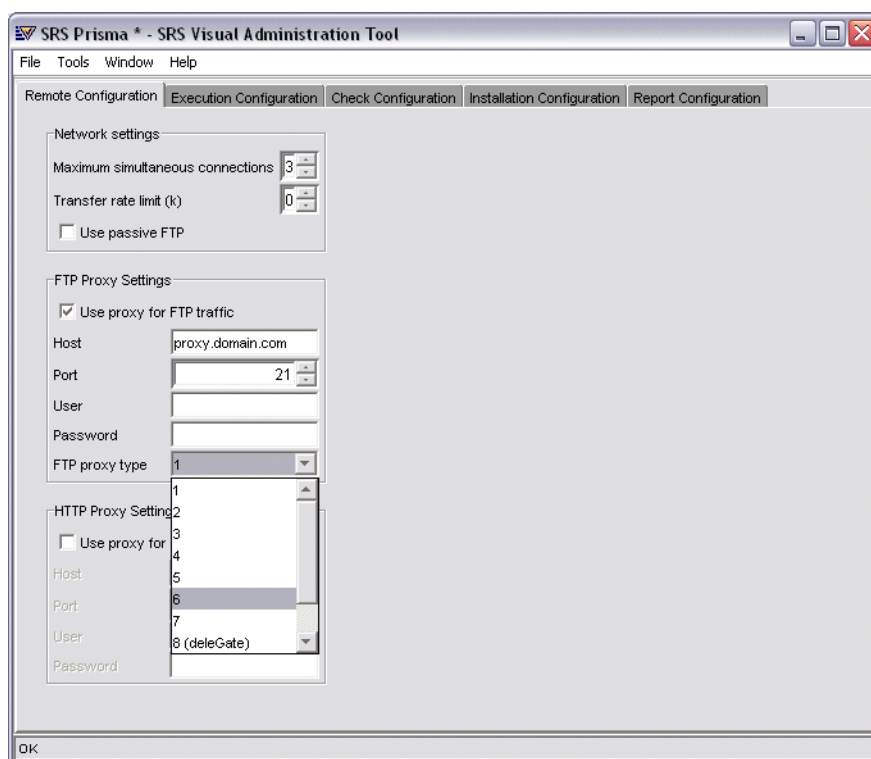


Figure 3.4 Configuring an FTP proxy

Host

This is the name of the host where the proxy is located.

Port

This is the port on which the proxy listens. By default, this is set to 21.

User

Some proxies require a username for use. This can be supplied here.

Password

Some proxies require a password for authentication. This can be supplied here.

FTP Proxy Type

SRS Prisma can deal with a large number of different types of proxies. These are identified by a type number between 1 and 10. For convenience, the number used for types 1-8 corresponds to the types used by the popular `ncftp` client. The types behave as follows:

1

FTP proxy where client logs into proxy with `user@host`

2

FTP proxy where client logs into proxy with proxy user/password followed by user login with `user@host`

3

FTP proxy where client logs into proxy, use `SITE host`, then remote username/password

4

FTP proxy where client logs into proxy, use `OPEN host` then remote username/password

5

FTP proxy where client logs into proxy with username `user@proxyUser@host`, password `pass@proxyPass`

6

FTP proxy where client logs into proxy with user `proxyUser@host`, password `proxyPass` followed by normal remote host login.

7

FTP proxy where client logs into proxy with user `user@host proxyusername`, remote password, then `ACCT proxypassword`

8

Use 'deleGate' proxy.

9

Use 'ftp.proxy' proxy

10

HTTP proxy. The proxy host supplied here is not an FTP proxy at all, but HTTP proxy and carries out FTP over HTTP. Note that this is not optimal, as the connections may be less robust and less easily recoverable than pure FTP connections.

3.2.2.3 Using an HTTP Proxy

SRS Prisma also supports the use of HTTP proxies for HTTP or HTTPS traffic. The proxy details can be set in the **Remote Configuration** tab:

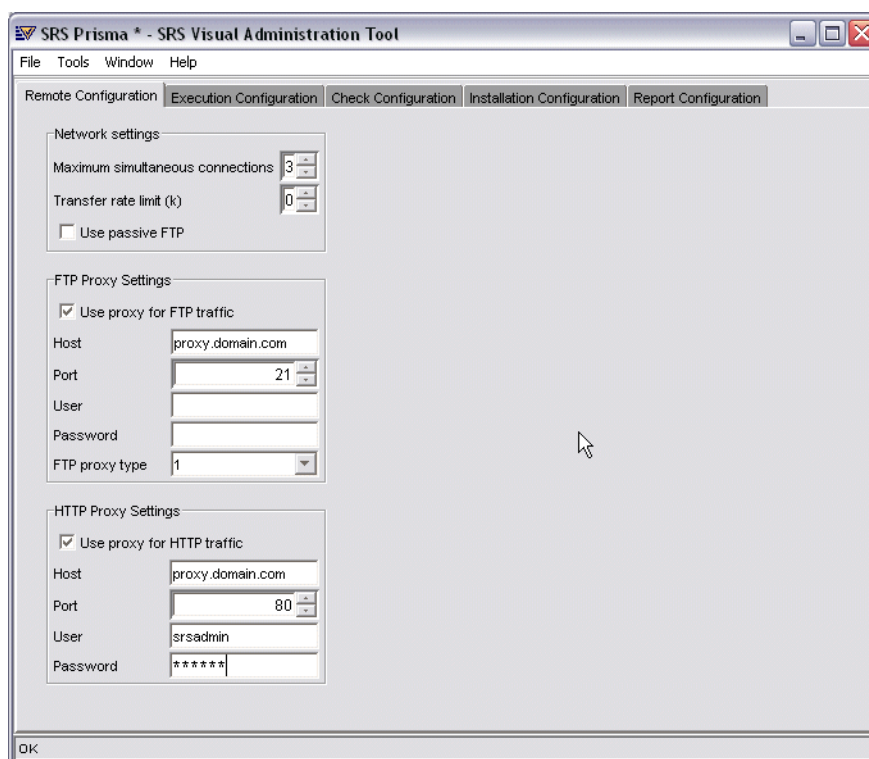


Figure 3.5 Configuring an HTTP proxy

Name

This is the name of the host where the proxy is located.

Proxy Port

This is the port on which the proxy listens. By default, this is set to 80.

Proxy Username

Some proxies require a username for use. This can be supplied here.

Proxy Password

Some proxies require a password for authentication. This can be supplied here.

3.2.2.4 Testing a Remote Connection

In order to test a remote connection from the SRS Prisma server after setting any required proxies etc., run:

```
% $SRSLION/prisma/testRemoteConnection.i
```

By default, this opens test FTP, HTTP and HTTPS connections to test sites at LION Bioscience. These defaults can be overridden using the following flags:

-ftpTarget

supply target URL for FTP connection

-httpTarget

supply target URL for HTTP connection

-httpsTarget

supply target URL for HTTPS connection

3.3 Execution Configuration

The **Execution Configuration** tab allows the fine-tuning of aspects of how SRS Prisma runs the commands necessary for updating.

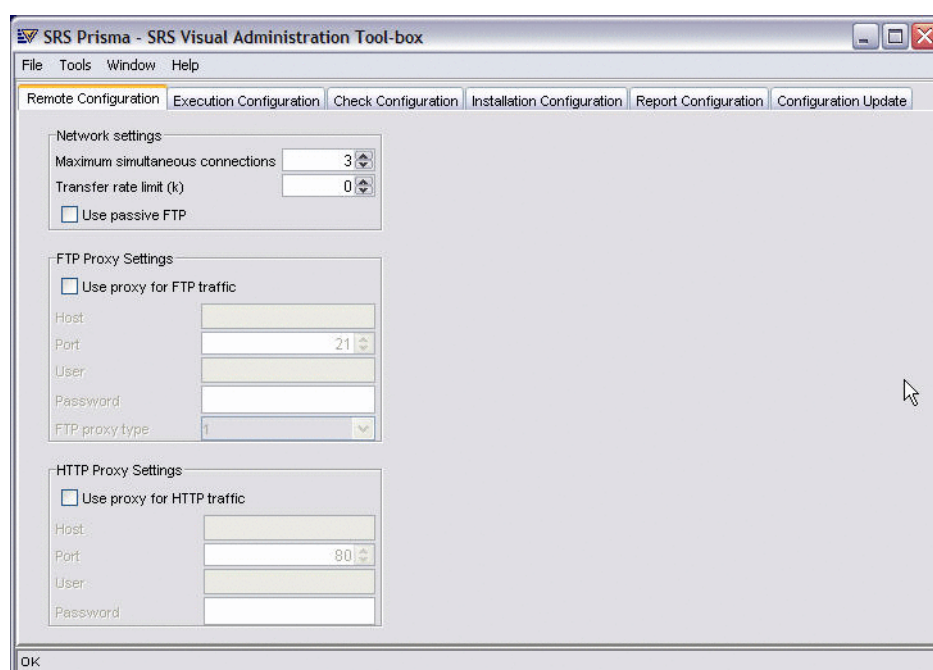


Figure 3.6 The Execution Configuration tab.

3.3.1 General Settings

3.3.1.1 Always honor PRISMA_RUNNING lockfile

The Prisma update process creates a lockfile (\$SRSPRISMA/PRISMA_RUNNING) when it starts, to prevent concurrent updates from occurring. However, if the update process is aborted, this file may not be removed, and will prevent. If the **Always honour PRISMA_RUNNING lockfile** is unchecked, SRS Prisma will remove this file if the process that created it cannot be found on the same host, and will start a new process.

3.3.1.2 Max. Repetitions of Failed Updates

Normally, SRS Prisma will attempt to run specified update commands once, and once only. However, SRS Prisma can be set to try to repeat failed updates up to a specified number of times. This applies to the update commands specified for each 'level' of libraries and links. This can be useful in a situation where transient hardware or network problems can lead to sporadic failures.

3.3.1.3 Make Command

By default, SRS Prisma uses GNU `make` (supplied as part of SRS) to execute makefiles containing update commands. However, you may wish to use another compatible make for this e.g. `ppmake` for distributed execution. The path to this make can be set here.



Note: Although other makes may work, SRS Prisma is only tested using the supplied version of GNU make.

3.3.1.4 Maximum Parallel Processes

Each make process can spawn multiple parallel execution processes. For a multiple processor machine, or a batch queue system, this number can be set to an appropriate level to maximize the efficiency of the make process. If 0 (zero) is specified, all processes will be launched in parallel. This is not normally recommended as the number of processes for a large update may well exceed system limits.

3.3.1.5 Shell Command

`/bin/sh` is normally used as the shell command used by make to execute targets, and as the command used to execute unpack and reformat commands etc. This can be changed to an alternative, POSIX-compliant Bourne shell if required.

3.3.1.6 Suppress Warnings from srsbuild

Non-fatal parser warnings are normally displayed by `srsbuild` during the indexing process. Check this option to allow these warnings to be suppressed.

3.3.1.7 Parser Logging Level

If parsers are used that contain internal logging commands (see Flat-File Parsers, page 253 of the *SRS Advanced Administrator's Guide*), this level can be set to change the level of logging used.

3.3.2 Using a Batch Queue System

SRS Prisma has been designed to allow simple integration with batch queueing and load balancing systems so that time-consuming or CPU-intensive processes such as indexing can be 'farmed out' across other machines. The system that SRS Prisma uses is very flexible, but there are a number of important requirements:

- Ensure that all batch queue used hosts share a common SRS installation for which all required binaries exist.
- Ensure data files and indices are accessible from each host in the same location.
- Ensure that all hosts used for executing pretranslation and reformatting commands have all required binaries in the correct location, or using the correct path if specified. For example, the location of `gunzip` should be common on all hosts.
- Ensure that all batch queue hosts used for download commands can access remote FTP sites.
- To prevent problems with NFS latency, a sleep command can be used as a pre- or post-execution script.
- When using a batch queue system, the shell used by make must be a POSIX-compliant `/bin/sh` or `/bin/bash -posix` to ensure the process can be run fully in the background.

These fundamental requirements apply regardless of the system used.

SRS Prisma supports use of Sun's Grid Engine and Platform's LSF, although it is also possible to use other systems, providing they meet some basic requirements. The next sections discuss for each of these what type of command is used to submit jobs to the system.



Note: The commands provided are suggestions only. They may need alteration to run on a given installation, or that installation may need configuring for them to work. If you are unsure about aspects of these commands, please consult the appropriate documentation for the batch queue system.

Note: With the introduction of SRS Prisma 4.1.1, the batch queue command no longer requires an explicit call to `. ${SRSETC}/ prep_srs.sh`. The appropriate command is now inserted as part of the command to be executed.

3.3.2.1 Platform LSF

The following command for use with a typical cross-platform LSF installation, but can be readily altered to add further controls (required complexes, specified queue/host names and so on). However, note that these are suggested only, and not guaranteed to work on all installations.

```
bsub -L /bin/bash -J \"p\${subst / ,_,\${@}}\" -K -R \"srs\" \"%COMMAND%\"
```

In addition, the following `bsub` flags are recommended:

-L /bin/bash

executes commands using BASH rather than log-in shell

-K

(required) wait for dispatch and execution

-R <hosttype>

required when cross-platform execution used. Hosts should be set to required type as appropriate.

-J "<jobname>"

(suggested) identification of running jobs. In this instance, a makefile variable string is used which inserts the current makefile target name.

3.3.2.2 Sun Grid Engine

The following commands is suggested for use with a typical Sun Grid Engine installation, but can be readily altered to add further controls (required complexes, specified queue/host names, and so on).

```
qrsh -nostdin -cwd -now n -N \"p\$(subst /,_,\${@})\" \"%COMMAND%\"
```

The following `qrsh` flags are used in this command:

-nostdin

(required) detach execution of rsh from a terminal

-cwd

(required) execute the command in the same directory

-now n

(required) wait for execution until a slot becomes available

-N "<jobname>"

(suggested) identification of running jobs. In this instance, a makefile variable string is used which inserts the current makefile target name.

3.3.2.3 Other Systems

Other batch queue systems can be readily added, providing the following criteria are met:

- Submission commands must be able to accept a command string (quoting is recommended) rather than a script name.
- Command execution must take place in the same directory (`SRSPRISMA/<run-name>/flags`).
- The submission command **must** wait for execution to complete.
- The submission command must execute jobs using BASH or a POSIX-compliant Bourne shell.

3.3.2.4 Configuration of Batch Queue System

The batch queue system panel of the **Execution Configuration** tab allows the system used to be selected, and the default batch command used to be altered if required:

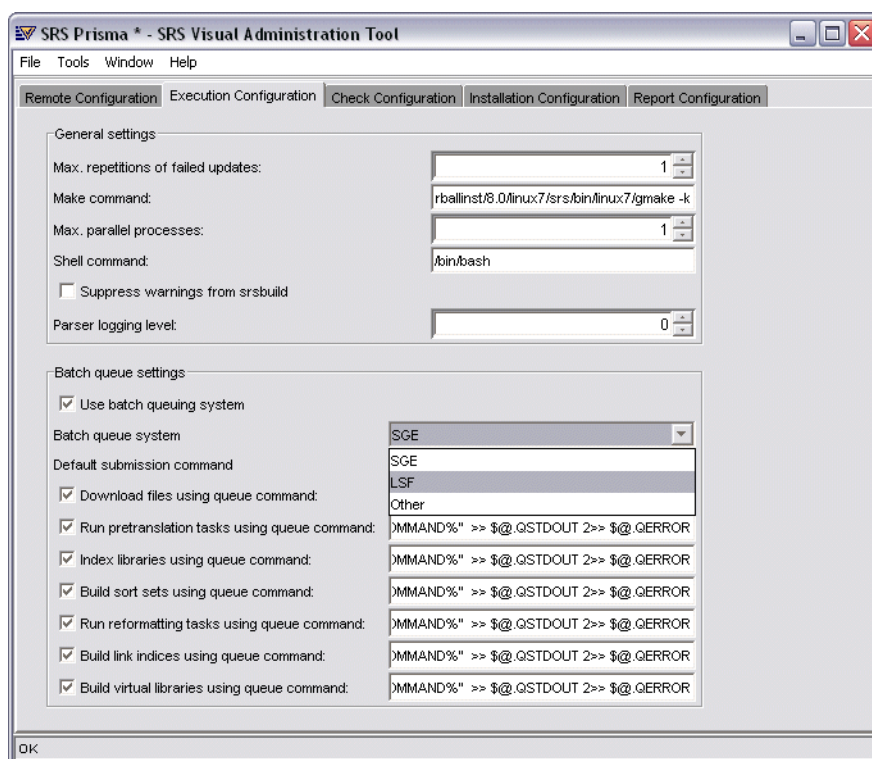


Figure 3.7 Configuring SRS Prisma to use a batch queue system.

In addition, each of the individual phases of the Prisma update process can be controlled so they can be excluded from the system, or dispatched using different commands (e.g. Different queues or resource requirements):

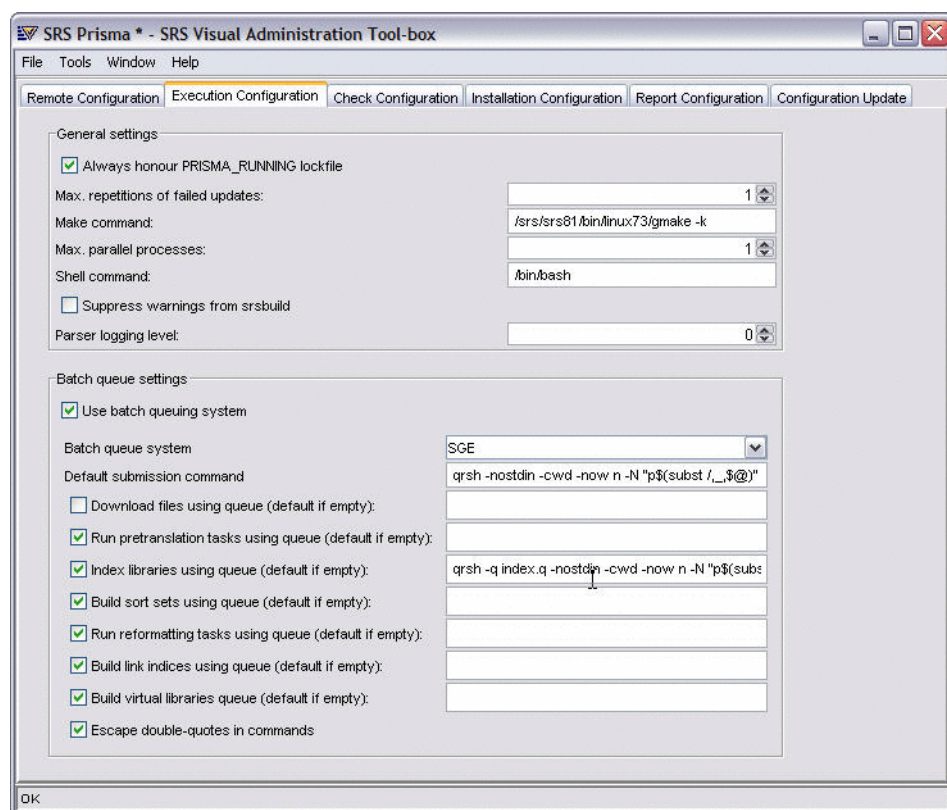


Figure 3.8 Controlling individual phases.

The following phases can be controlled individually:

- Download (all remote and local file downloads)
- Pretranslation (all unpacking and preprocessing commands)
- Index (all indexing jobs)
- Sort (all sort set creation jobs)
- Reformat (all FASTA/BLAST file generation and other reformatting jobs)
- Link (all link indexing jobs)
- Virtual (all virtual indexing jobs)

It is also possible to assign specific commands to these phases for specific libraries. See Section 3.13, Library-specific batch queue commands, page 118 for more detail.

A batch queue system can also be specified directly in the `$SRSETC/prisma/conf` file, using the following attributes of the `PrismaExecutionSettings` object:

useBatch

should be set to 'y' or 'n'

batchQueueSystem

should contain a `PrismaBatchQueue` object (see below)

The `PrismaBatchQueue` object has the following attributes:

type

batch queue system (LSF,SGE,Other)

defaultBatchCommand

default command for submission

downloadBatchCommand**pretransBatchCommand****indexBatchCommand****sortBatchCommand****reformatBatchCommand****linkBatchCommand****virtualBatchCommand**

commands for specific stages to use instead of the default

useBatchForDownload**useBatchForPretrans****useBatchForIndex****useBatchForSort****useBatchForReformat****useBatchForLink****useBatchForVirtual**

set to 'y' or 'n' depending on whether stage should use commands

escapeQuotes

escape quotes automatically (see Section 3.3.2.5, Batch Queuing and Quote Escaping, page 41).

3.3.2.5 Batch Queuing and Quote Escaping

The requirement for double quoting of commands means that all double quotes used in unpack and reformat commands must be escaped. SRS Prisma can do this automatically by setting the **Escape quotes** option (this is done by default when using batch queuing).

3.3.2.6 Testing a Batch Queue System

The settings used for your batch queue system can be tested using the following command:

```
% $SRSLION/prisma/testBatchQueue.i
```

This will use all specified commands to run a simple shell command to report the version of srsbuild used. This command can be overridden using the command line flag `-testCommand`.

3.4 Check Configuration

SRS Prisma normally checks all libraries both remotely and locally. For experienced users, the extent of this checking process can be controlled using the Check Settings Tab:

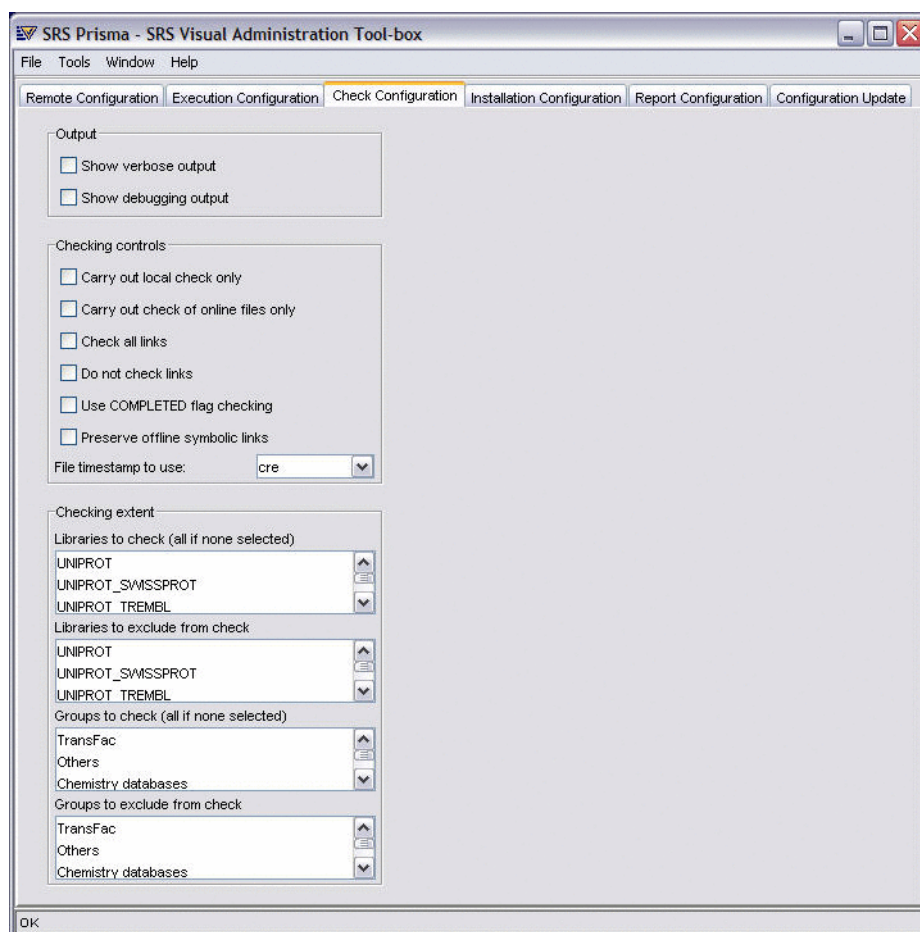


Figure 3.9 The Check Configuration tab.



Note: Changing these settings may have unexpected consequences for the consistency of your installation. Please ensure you understand the role of each setting before altering it.

The following aspects can be controlled:

3.4.1 Show Verbose Output

This option turns on verbose output of the check phase by default.

3.4.2 Show Debugging Output

This option turns on diagnostic output of the check phase by default (this is only usually of use for debugging by LION support personnel).

3.4.3 Carrying Out Local Check Only

To prevent SRS Prisma from automatically checking and downloading new files from remote servers, this option can be set.

3.4.4 Carry Out Check of Online Files Only

Do not check any offline files and carry out all updating online.

3.4.5 Check All Links

Check and build all links both between selected libraries and all other libraries on the installation.

3.4.6 Do Not Check links

Do not build any links between selected libraries.

3.4.7 Use COMPLETED Flag Checking

By setting this option, SRS Prisma can control checking of local files by looking for the presence of a `Completed` flag in the offline data directory before allowing indexing and installation of offline data. This flag file is used to signal that an external download process to the offline directory has completed.



Note: This mechanism only applies to libraries for which the attribute `completedFile` of the `Library` object has been set to 1.

3.4.8 Preserve Offline Symbolic Links

Normally, SRS Prisma removes any symbolic links found in offline data directories as these may be old links remaining from previous aborted runs, and moving them online might result in circular links. However, it is sometimes necessary to index using a symbolic link. In this case, select this option to suppress automatic removal of these links. In this case, it is recommended that offline directories be manually checked after failed runs to ensure unwanted links are removed before the next update.

3.4.9 File Timestamp to Use

By default, SRS Prisma compares local files to each other and to indices by using the creation date from the file time stamp. However, some backup systems touch this date, and it may be more advisable to examine the modification date instead. This option can be set to `cre` (default), `mod` or `acc` to control the local time stamp used.



Note: This does not influence the time stamp used for remote file checking.

3.4.10 Libraries to Check

A list of library names can be supplied (as a space-separated list if using the command line tool) to check specifically (others will not be checked). If left blank, all libraries will be checked.

3.4.11 Libraries to Exclude from Checking

As above, except these libraries will explicitly be excluded from checking.

3.4.12 Groups to Check

As above, but a list of groups can be supplied instead of libraries.

3.4.13 Groups to Exclude from Checking

As above, except these groups will explicitly be excluded from checking.

3.5 Installation Configuration

3.5.1 Pre-installation Checks

SRS Prisma normally installs all new data and indices into their online location once all stages of updating have completed. However, it is possible to specify one or more conditions that must be fulfilled before this can take place. SRS Prisma will then wait, testing all specified conditions every 5 minutes (by default) until all are satisfied. One or more of the following can be used.

Specified time window

A string representing a window of time can be specified. Installation will only take place within this window. A window can be specified using a string of the format d:hh:mm-d:hh:mm where d represents the day of the week where 0 (zero) represents Sunday (optional), and hh:mm represents a time in the 24h clock. For instance, 03:00-05:00 means daily between 3-5am, whilst 5:17:00-1:08:00 represents between Friday 6pm and Monday 8pm. Multiple windows can be specified as a comma-separated list.

Specified flag file

A file can be specified that must be present before installation can proceed.

Specified command

A shell command can be specified. This must return with an exit status of zero before installation will proceed.

The conditions can be set using the **Installation Configuration** tab:

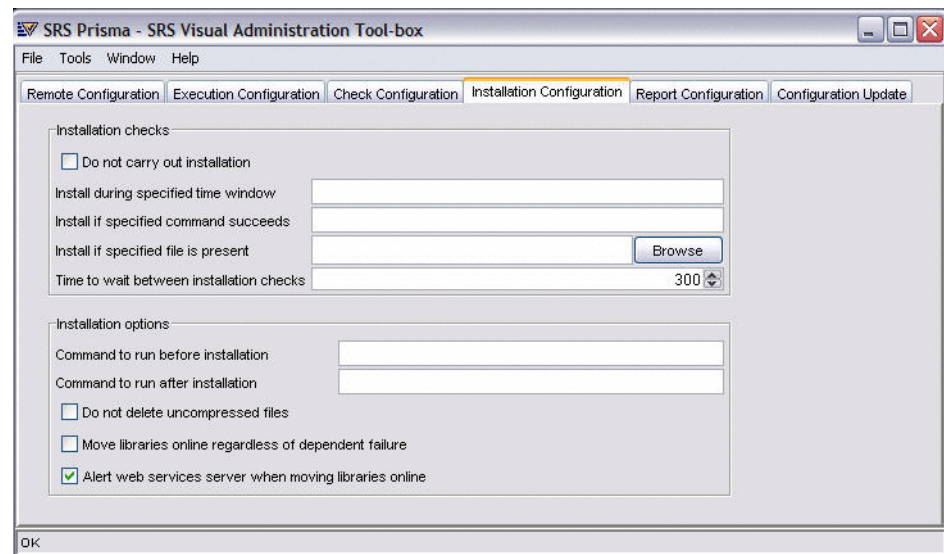


Figure 3.10 The Installation Configuration tab.

In addition, the period to wait between checking these conditions can also be set.

3.5.2 Installation Options

Additional options can be used to control installation and can be set using the **Installation Configuration** tab.

3.5.3 Command to Run Before Installation

This command will be executed immediately before the installation process starts. This might be used to stop an associated server.

3.5.4 Command to Run After Installation

This command will be executed after the installation process completes. This might be used to restart or refresh an associated server.

3.5.5 Do Not Delete Uncompressed files

If SRS compression is used (see Chapter 14, Data Compression, page 223 of the *SRS Advanced Administrators Guide*) then normally uncompressed files are removed during the installation phase. This option can be used to preserve the original files, e.g. during testing.

3.5.6 Move Libraries Online Regardless of Dependant Failure

As described in Section 1.2.3.4, Step 5: Block Failed Libraries and move Complete Libraries, page 7, the failure of a dependent library or virtual library will block the installation of all members/parents in order to preserve consistency of data. This global flag blocks this behavior for all libraries. Note that using this option may lead to loss of consistency between libraries. Additionally, it may be preferable to suppress this behavior individual libraries instead (see Section 3.11.4.1, Hierarchy Checking, page 109).

3.6 Report Configuration

3.6.1 HTML Report Generation

SRS Prisma produces detailed HTML reports following each update, and these are described in detail in Chapter 5, of this Guide. However, the contents of these reports can be tailored to suit individual needs using the **Report Configuration** tab:

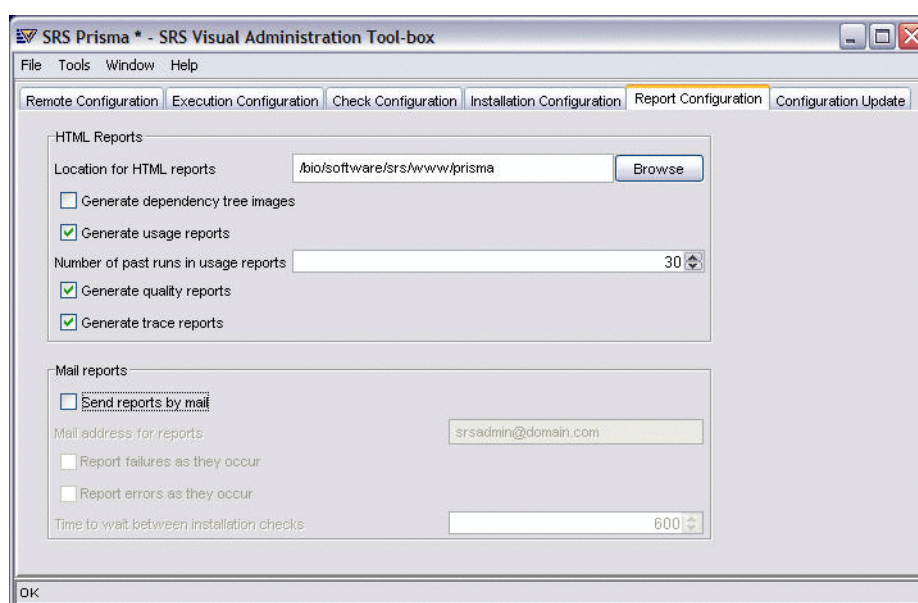


Figure 3.11 The Report Configuration tab.

3.6.2 Location for HTML Reports

Normally, HTML reports are written to the `$SRSWWW/prisma` directory tree, but this can be altered to any location. Note that if this location is changed, then you must ensure that the webserver is still able to serve this information. It is suggested that this is done by symbolically linking this directory to `$SRSTAGS/web/prisma` so that it is available as `http://host/srs/prisma/`

3.6.3 Generate Dependency Tree Images

It is possible for SRS Prisma to generate complex dependency tree diagrams showing the relationships between different tasks in an update, and the relative CPU time spent on each task (described further in Section 5.3.13, The Dependency Tree, page 176). This is time-consuming and is normally turned off, but can be reactivated with this option.

3.6.4 Number of past runs in usage report

SRS Prisma can store details of resource usage (e.g. CPU time, disk space etc.) associated with an update process over the course of multiple runs in a Usage Report (see Section 5.3.15, The Usage Report, page 184). By default, the last 30 runs are shown. This option allows the number of stored runs to be changed.

3.6.5 Generate Quality Reports

SRS Prisma normally finishes each run by running a full quality report on all libraries. If this is not required, it can be turned off with this option. Quality Reports are discussed in Chapter 7, SRS Prisma Quality Report.

3.6.6 Generate Trace Reports

Normally, SRS Prisma generates a full data and index file listing before and after each update. However, if this is not required, it can be turned off with this option. Trace reports are discussed in Section 5.3.14, The Decision Report, page 178.

3.6.7 Mail-based Reporting

In addition to the detailed HTML reports generated during a Prisma run, SRS Prisma can also generate short text reports and mail them to the administrator. Reports can also be sent when new failures and error occur (note that this requires a functional sendmail on the main Prisma host). This can be set using the Report Configuration tab.

Send reports by mail

Check this option to turn on mail-based reporting.

Mail address for reports

This is the e-mail address that reports are sent to.

Report failures as they occur

When this option is active, new failures are checked for on a regular basis and email alerts sent if they are found.

Report errors as they occur

When this option is active, new errors (non-fatal) are checked for on a regular basis and email alerts sent if they are found.

Time to wait between checks

This option allows control of how often new failures and errors are checked for.

3.6.8 Quality Reports

After the update process is complete, SRS Prisma runs a selection of tests on the installed libraries to locate and flag any errors. The results are presented in a Quality Report (see Chapter 7, SRS Prisma Quality Report). Some aspects of this report are

configurable, and can be set using the Report Configuration tab. The following properties may be configured:

Generate quality reports

This option controls whether quality reports are automatically generated as part of the main SRS Prisma update process.

Number of entries to test

The quality tests involve checking a selection of entries from each library. This option controls how many entries are checked.

Time out for parser tests

This option controls the time out (in seconds) for tests involving parsers.

Run tests on tools

By default, all installed tools are tested. However, this can extend the time required, particularly if EMBOSS is installed. This option can be used to control whether tools are tested.

Test suites to check

Test suites to exclude from check

Normally, all test suites are checked. These options allow more precise specification of which test suites are checked.

3.7 Configuration update

SRS Prisma 4.1 is capable of automatically updating meta-data files for SRS configuration. This feature can be configured with the **Configuration Update** tab as shown in Figure 3.12:

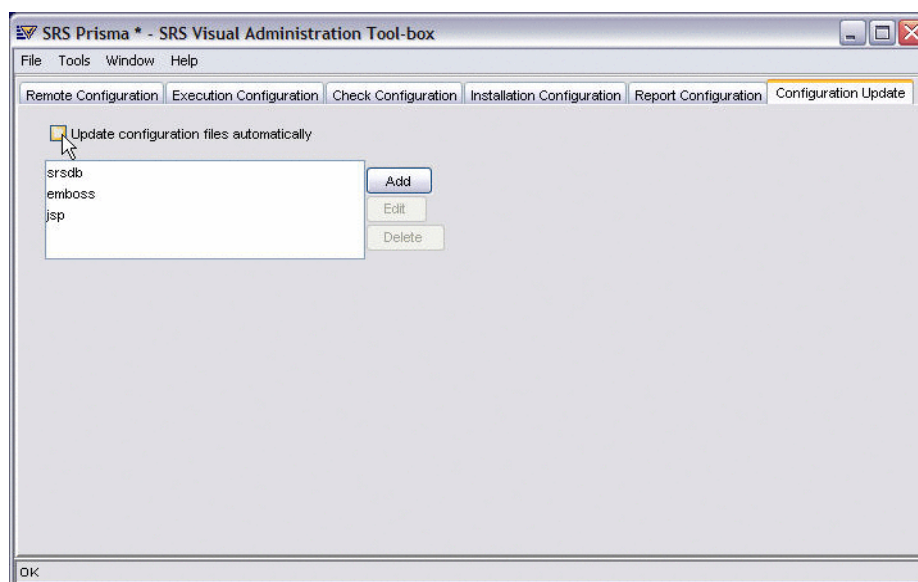


Figure 3.12 The Configuration Update tab.

The use of this configuration tool is described in detail in Chapter 8, Configuration Updates with SRS Prisma.

3.8 Configuring a Library for Use with SRS Prisma

The SRS Prisma module of VisAd is designed to allow easy configuration of libraries for use with Prisma, and it is recommended for all users. To open a library for editing, launch the Prisma tool from the main window icon or the **Session** menu, open the **Tools** menu and select **Edit library**. A dialog then appears where the library name can be selected:

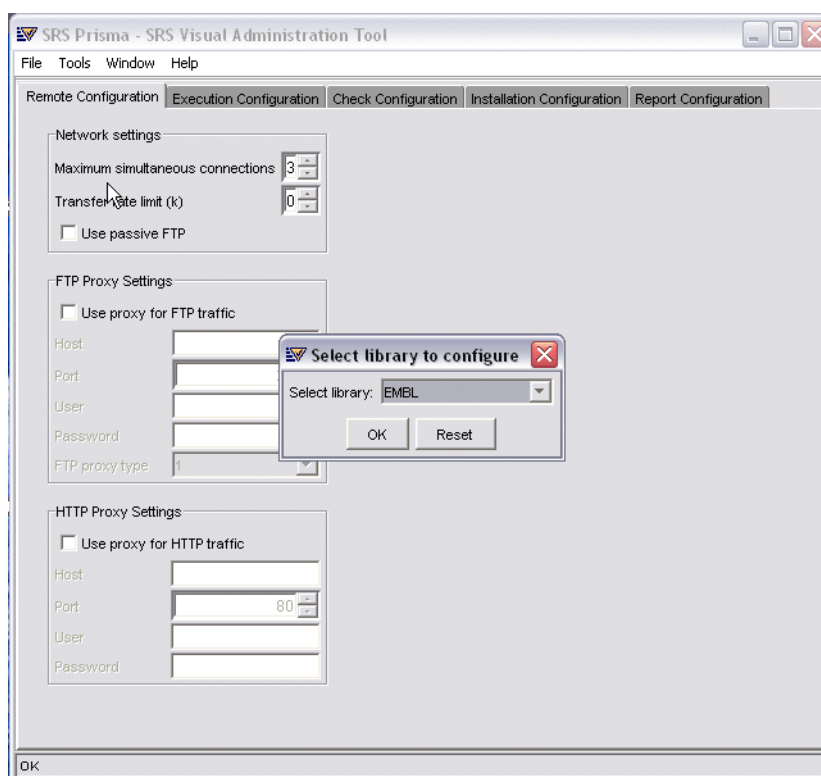


Figure 3.13 Opening a library for editing with the Prisma tool.

Selecting the appropriate library and clicking 'OK' opens the Library Configuration dialog. This is divided into 5 panes:

- Update settings (not virtual libraries)
- Indexing settings (not virtual libraries)
- Reformat settings
- Installation settings
- Schedule settings

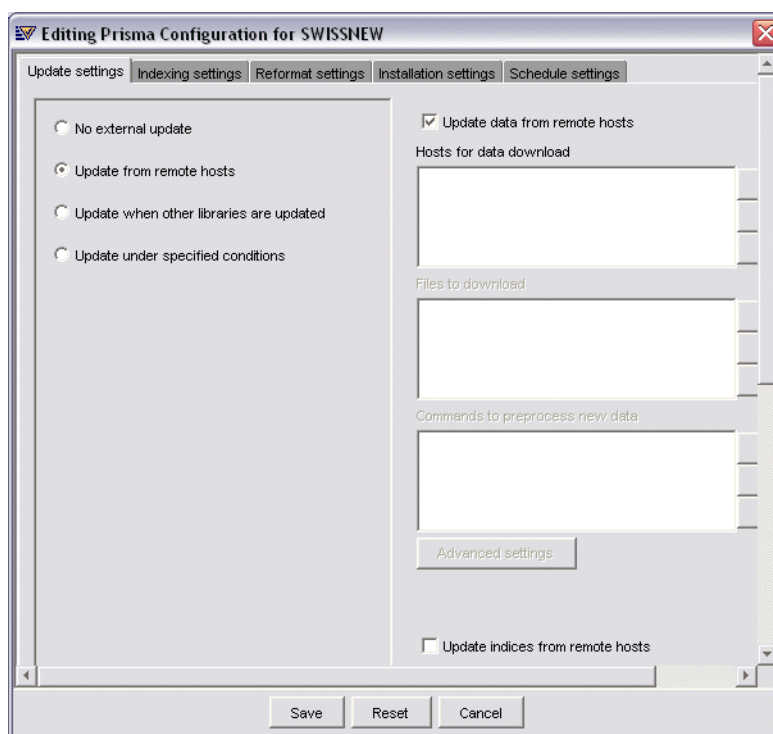


Figure 3.14 The Library Configuration dialog.

The rest of this guide refers to these panes directly with more information on how to use them to configure different aspects of a library for use with SRS Prisma.

Alternatively, experienced users who are familiar with the syntax of Icarus may wish to edit Icarus files directly. The central class used by SRS Prisma is the `Resource` class, and most of the configuration in this guide deals with different attributes and sub-classes of this class, and describes which attributes should be altered. Note that this is not recommended unless the user is comfortable working with Icarus classes and objects.

The `Resource` object is generally stored in a file with the name of the library (in lower case) and the extension `.it` e.g. the `Resource` object for EMBLNEW is stored in `SRSDB:emblnew.it`. By convention, the instance of the `Resource` class is assigned to a variable based on the original library e.g.

Example 3.1 The Resource object for EMBLNEW stored in SRSDb:emblnew.it

```
$EMBLNEW_Res=$Resource:[
  name: EMBLNEW
  updMethod: mirror
  remoteHosts: $RemoteHost:[
    hostName: "ftp.ebi.ac.uk"
    port: 21
    downloadDir: "/databases/embl/new/"
  ]
  remoteFiles: $RemoteFilePattern:[
    searchPatternRemote: 'cumulative.dat.gz'
    fromNamePattern: '.gz' toNamePattern: ''
  ]
  unpackCommand: "$gunzip %s.gz"
]
```

The Resource object is then referenced in the corresponding Library object using the `res` attribute e.g.

Example 3.2 The Library object for EMBLNEW stored in SRSDb:emblnew.i

```
$EMBLNEW_DB=$Library:[
  name: EMBLNEW
  parallelType: fileSize
  printName: 'EMBL (Updates)'
  group: $SEQUENCESUB_LIBS
  format: $EMBL_FORMAT
  ...
  res: $EMBLNEW_Res
  searchName: '*.dat' loaderCache: $DnaDBInfo_Class concepts: nucleicAcidSeq
  isPartOf: EMBL
]
```

3.8.1 Configuring a Library for Remote Update

The first step in activating remote updates for a new library is to indicate that the library is updated remotely. From within VisAd this is done by opening the library for editing, selecting the **Update Settings** tab and then selecting **Update from remote hosts**.

From within the `Resource` object, the equivalent action is to set the attribute `updMethod` to `mirror` e.g.

```
$EMBLNEW_Res=$Resource:[EMBLNEW  
  updMethod:mirror  
  ...  
]
```

3.8.1.1 Remote Data Updates

3.8.1.2 Local Locations

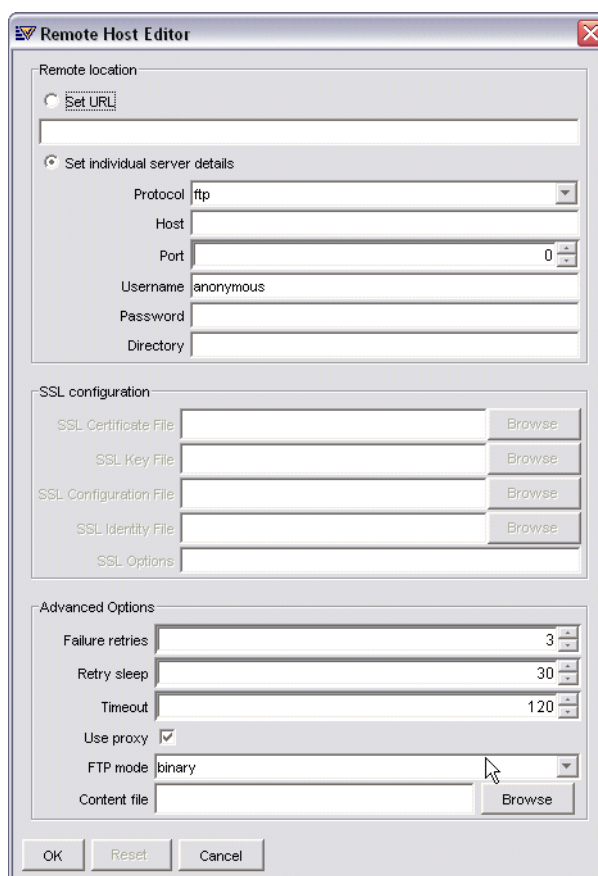
If remote updates are activated, it is essential that the library has both an online and offline directory that exist and are writable. Consult Chapter 9, Adding Flat-File Libraries, page 119 of the *SRS Advanced Administrators Guide* for more details about setting these directories.

3.8.1.3 Specifying Remote Locations

In order to download data to update a library, you must specify at least one location from which the data may be retrieved. This must specify the protocol to use, host-name and port if applicable, path to the directory where the data is stored and any authentication information needed. This may be specified as a URL or as individual values.

If multiple hosts are defined, then they are assumed to be mirrors of each other to allow failover, and will be used in order of preference.

To add a new host using VisAd, ensure that both **Update data from remote hosts** checkboxes are selected, and click **Add** on the panel labeled **Hosts for data download**. This opens a new dialog where details may be added individually or as part of a URL (described below). Click **OK** to save the new host to the list for the current library. Hosts can be edited or deleted later using the same list.



The **Remote Host Editor** dialog box is used to configure a remote host. It features three main sections: **Remote location**, **SSL configuration**, and **Advanced Options**. The **Remote location** section has two radio buttons: **Set URL** (unselected) and **Set individual server details** (selected). Below the radio buttons are fields for **Protocol** (set to **ftp**), **Host**, **Port** (set to **0**), **Username** (set to **anonymous**), **Password**, and **Directory**. The **SSL configuration** section includes five fields with **Browse** buttons: **SSL Certificate File**, **SSL Key File**, **SSL Configuration File**, **SSL Identity File**, and **SSL Options**. The **Advanced Options** section contains **Failure retries** (set to **3**), **Retry sleep** (set to **30**), **Timeout** (set to **120**), a checked **Use proxy** checkbox, **FTP mode** (set to **binary**), and a **Content file** field with a **Browse** button. At the bottom are **OK**, **Reset**, and **Cancel** buttons.

Figure 3.15 Editing a remote host.

To add a new host directly using the `Resource` object, a `RemoteHost` object needs to be added to the `remoteHosts` list attribute of the `Resource` object. The details may be added as a URL, or as individual details (described below).

```
$UNIPROTSWISS_Res=$Resource:[...  
  updMethod:mirror  
  remoteHosts:$RemoteHost:[  
    hostName:"ftp.uniprot.org"  
    port:21  
    downloadDir:"/pub/databases/uniprot/knowledgebase/"  
  ]  
  remoteFiles:$RemoteFilePattern:[  
    searchPatternRemote:'uniprot_sprot.dat.gz'  
    fromNamePattern:'.gz' toNamePattern:''  
  ]  
  unpackCommand:"$gunzip %s.gz"  
]
```

3.8.1.4 FTP

Remote locations using FTP can be specified as a URL of the form `ftp://user:password@host:port/path/to/dir` (either using the URL portion of the Remote Host dialog, or as the `url` attribute of the `RemoteHost` object) e.g.

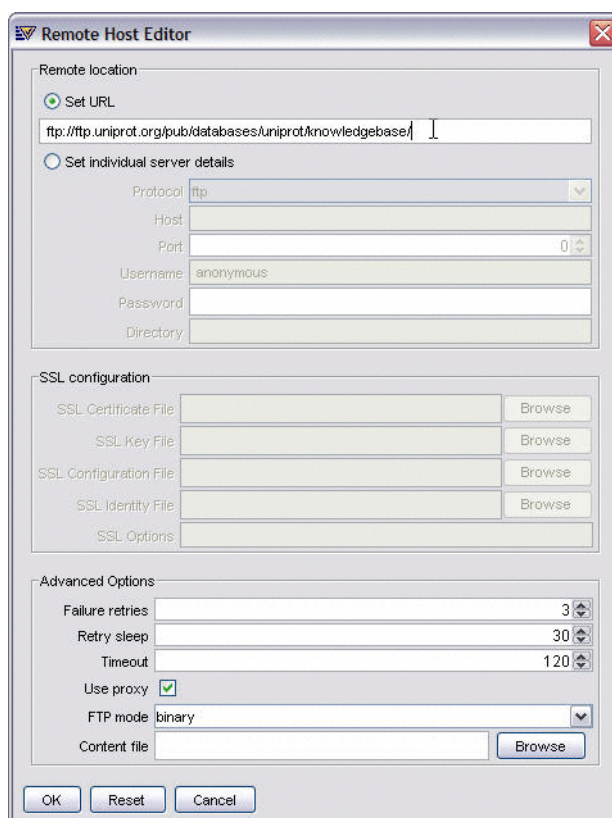


Figure 3.16 Setting an FTP host using a URL.

```
$RemoteHost: [url:"ftp://ftp.uniprot.org/pub/databases/uniprot/knowledgebase/"]
```

Alternatively, the host, port, download directory, username and password can be specified separately.

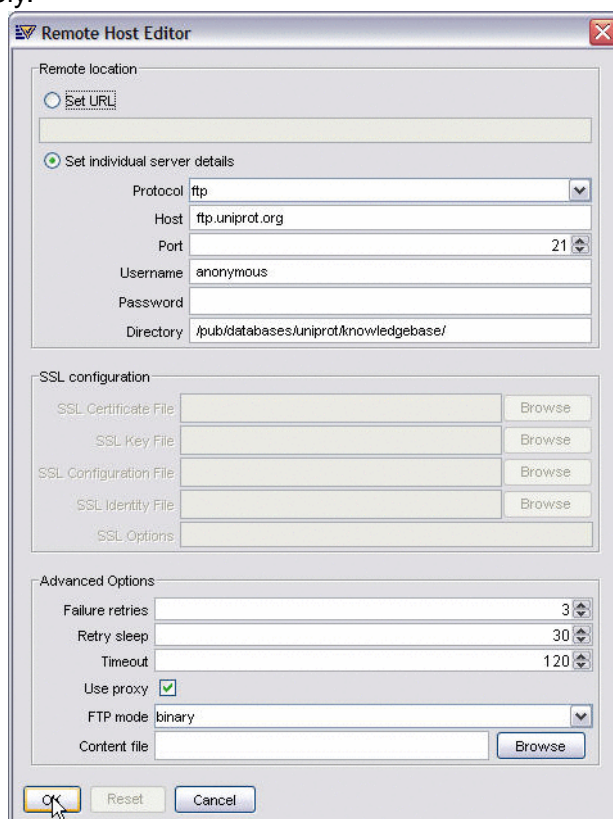


Figure 3.17 Setting an FTP host using individual properties.

```
$RemoteHost: [
  hostName:"ftp.uniprot.org"
  port:21
  downloadDir:"/pub/databases/uniprot/knowledgebase/"
]
```

Note: If username and password are not specified, the username `anonymous` will be used, and the email address used during SRS installation will be used for login. If the port is left empty, port 21 will be used.

In addition, the FTP transfer mode can be set to ASCII or binary by setting the **FTP mode** drop-down menu appropriately, or by setting the `transferMode` attribute of the `RemoteHost` object to `binary` or `ascii`.

```
$RemoteHost:[
  hostName:"ftp.uniprot.org"
  port:21
  downloadDir:"/pub/databases/uniprot/knowledgebase/"
  transferMode:ascii
]
```

Normally, all files within a directory are listed directly. However, sometimes it is desirable to use a listing read from a remote file instead. The location of this file can be specified using the **Content file** entry box, or as the `contentFile` attribute of the `RemoteHost` object:

```
$RemoteHost:[url:"ftp://ftp.uniprot.org/pub/databases/uniprot/knowledgebase/"
  contentFile:"pub/databases/uniprot/knowledgebase/LIS"
]
```

3.8.1.5 HTTP

HTTP hosts are specified in the same way as FTP hosts, either entering details directly or specifying a URL e.g.

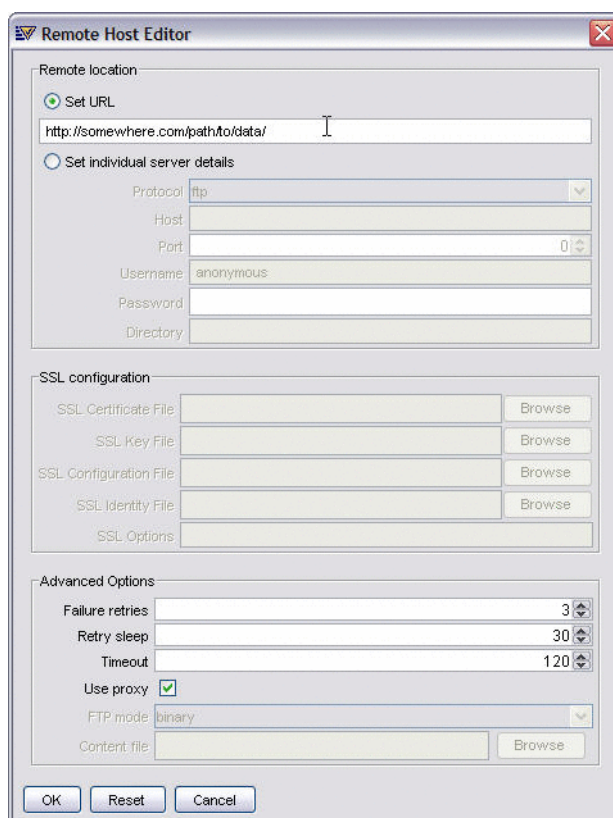


Figure 3.18 Specifying an HTTP host.

```
$RemoteHost: [url:"http://somewhere.com/path/to/data/"]
```

Note that transfer mode and contentFile are no longer appropriate, and that the default port is 80.

3.8.1.6 HTTPS

HTTPS hosts are specified in the same way as HTTP hosts, and in addition, an SSL certificate file and certificate key file can be specified for certificate-based client authentication. Note that these cannot be passphrase encrypted as Prisma does not support use of passphrases of this kind during the download process. These can be specified using the appropriate portions of the Remote Host dialog, or with the `sslCertFile` and `sslCertKey` attributes of the `RemoteHost` object:

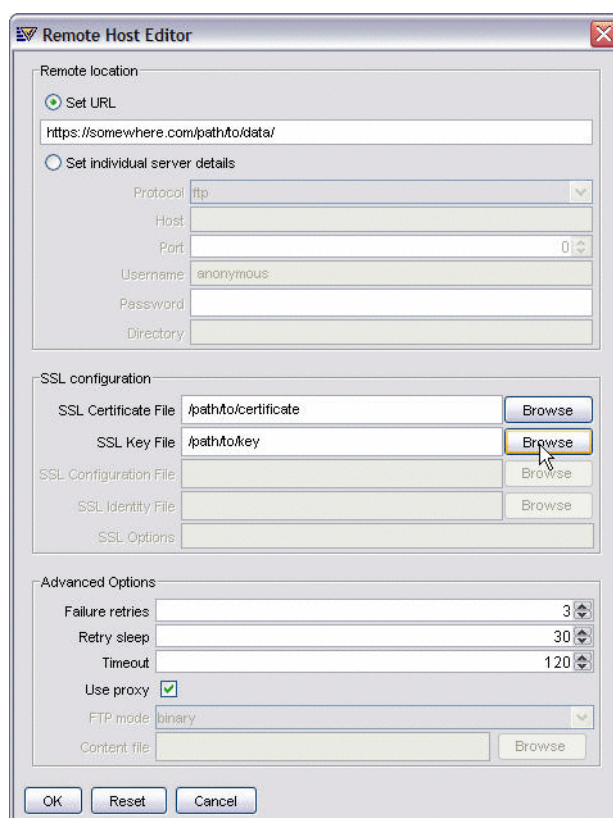


Figure 3.19 Specifying an HTTPS host.

```
$RemoteHost: [
  url:"https://somewhere.com/path/to/data/"
  sslCertFile:"/path/to/certificate"
  sslKeyFile:"/path/to/key"
]
```

3.8.1.7 File

If the desired files are located on a locally accessible file system, a Remote Host of type **file** can be used to check and copy them to the SRS installation. In this instance, the only applicable detail is the download directory although this can also be specified as part of a URL-style string of format `files:/path/to/files`.

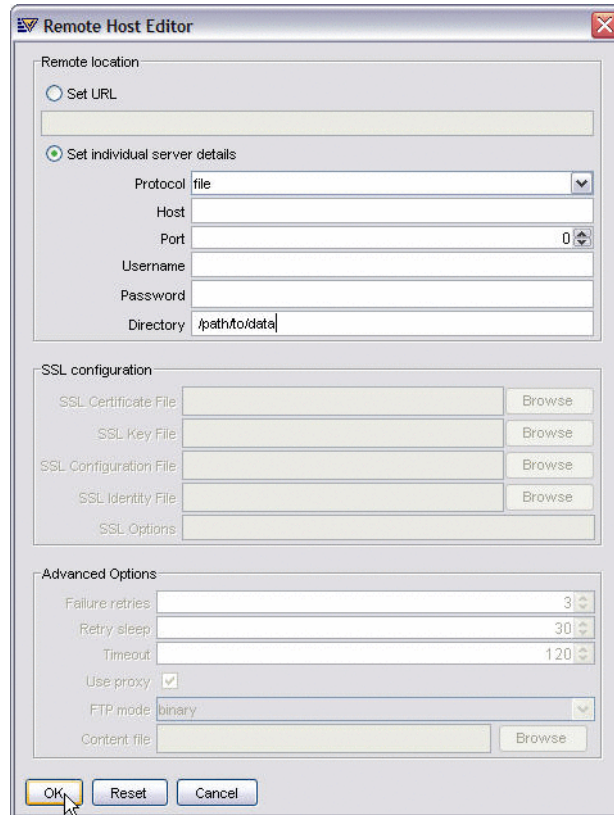


Figure 3.20 Specifying a local 'file' host.

```
$RemoteHost: [url:"file:/path/to/data/"]
```

3.8.1.8 SCP

Files can also be securely copied from a remote host using combination using the SSL tools `ssh` and `scp` (not supplied – tested for use with OpenSSL). `ssh` is used to list the files on the remote host, and `scp` is used to copy the desired files to the SRS installation.

This can be achieved with a Remote Host of type **scp**. The hostname and path can be supplied as individual details, or as a URL-style string of the form `scp://host/path/to/files/`.

Additionally, the SSL identity file and configuration file to use with `ssh/scp` can be supplied, and any additional options to be passed to `ssh/scp` can also be set:

The screenshot shows the 'Remote Host Editor' window. It has three main sections: 'Remote location', 'SSL configuration', and 'Advanced Options'. In the 'Remote location' section, 'Set individual server details' is selected. The 'Protocol' is 'scp', 'Host' is 'moran', 'Port' is '22', 'Username' and 'Password' are empty, and 'Directory' is '/tigris/'. The 'SSL configuration' section has four rows: 'SSL Certificate File' with path '/path/to/certificate', 'SSL Key File' with path '/path/to/key', 'SSL Configuration File' with path '/home/watson/.ssh/config', and 'SSL Identity File' with path '/home/watson/.ssh/identity.pub'. Each has a 'Browse' button. 'SSL Options' is 'Compression=yes'. The 'Advanced Options' section has 'Failure retries' (3), 'Retry sleep' (30), 'Timeout' (120), 'Use proxy' (checked), 'FTP mode' (binary), and 'Content file' (empty) with a 'Browse' button. At the bottom are 'OK', 'Reset', and 'Cancel' buttons.

Figure 3.21 Specifying a 'scp' host.

```
$RemoteHost: [protocol:scp  
  hostName:"moran"  
  downloadDir:"/tigris/"  
  sslIdentityFile:"/home/watson/.ssh/identity.pub"  
  sslConfigFile:"/home/watson/.ssh/config"  
  sslOptions:"Compression=yes"
```

```
]
```



Note: SSL keyboard-based authentication cannot be used with SRS Prisma as there is no facility for supplying this in batch.

3.8.1.9 General Options

Other general options can be specified:

Use proxy

Whether to use the appropriate proxy when dealing with this remote host.

Specified as the `useProxy` attribute of the `RemoteHost` object (default value is 'yes').

Failure retries

Number of times to retry if connection fails. Specified as the `mirrorRetry` attribute of the `RemoteHost` object (default value is 3).

Retry sleep

Time to wait (in seconds) after failed connection attempt before retrying.

Specified as the `retrySleep` attribute of the `RemoteHost` object (default value is 30 s).

Timeout

Time in seconds to wait before dropping dead connection. Specified as the `mirrorTimeout` attribute of the `RemoteHost` object (default value is 120 s).

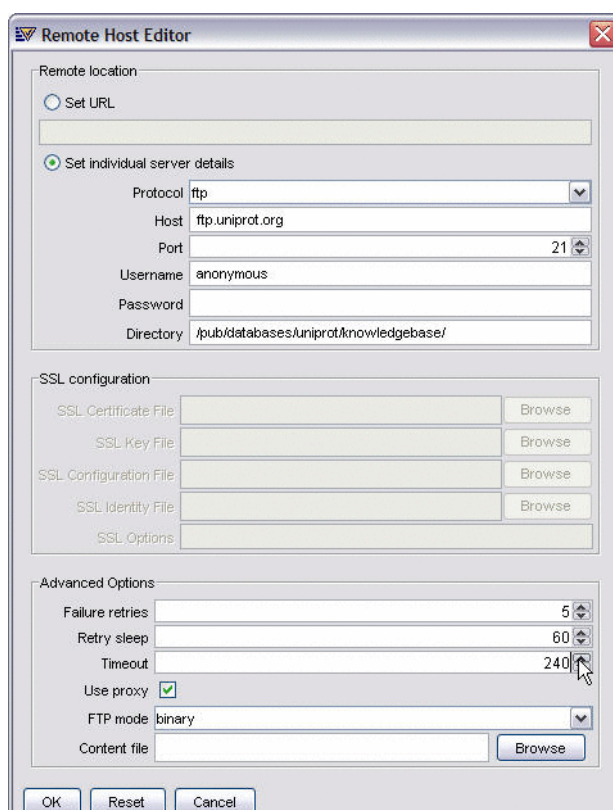


Figure 3.22 General remote options.

```
$RemoteHost: [  
  url:"ftp://ftp.uniprot.org/pub/databases/uniprot/knowledgebase/"  
  useProxy:n retrySleep:60  
  mirrorTimeout:240 mirrorRetry:5  
]
```

3.8.1.10 Specifying Remote Files

In addition to specifying one or more remote hosts, one or more remote file patterns must be specified. Each remote file pattern contains a regular expression for files to match on the remote site, a regular expression for files to match on the local site, and a list of rules on how to convert from the remote file name to the local name.

To add a remote file pattern, click on the **Add** button of the **Files to download** panel to open a **Remote File Editor** dialog box for editing these details:

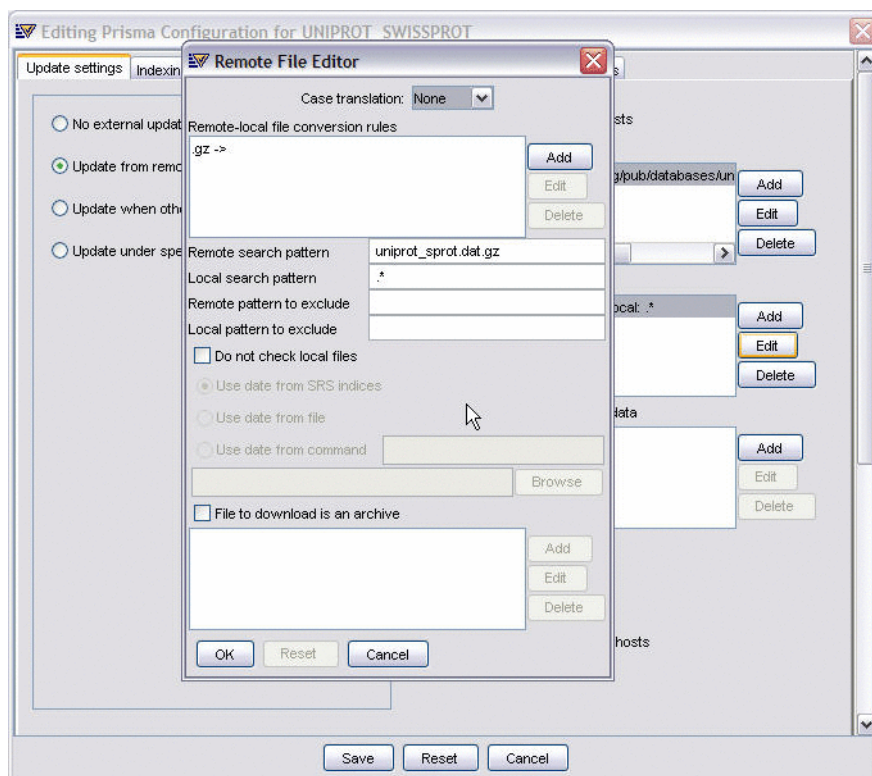


Figure 3.23 Specifying a remote file.

Alternatively, a new `RemoteFilePattern` object can be added to the `remoteFiles` attribute of the `Resource` object:

```
$UNIPROTSWISS_Res:[..
  remoteFiles:{
    $RemoteFilePattern:[
      searchPatternRemote:'uniprot_sprot.dat.gz'
      fromNamePattern:'.gz' toNamePattern:''
    ]
  }
]
```

The files on the remote site are specified using a regular expression, either entered into the **Remote search pattern** box or set as the `searchPatternRemote` of the `RemoteFilePattern` object. Likewise, the corresponding files on the local site are specified using a regular expression, either entered into the **Local search pattern** box or set as the `searchPatternLocal` attribute of the `RemoteFilePattern` object.

In addition, for directories where only a subset of files are required, the `searchPatternRemoteExclude` and `searchPatternLocalExclude` attributes of the `RemoteFilePattern` object can be used. These are used to specify regular expressions matching files that should not be checked, and are useful where a single regular expression is not sufficient to specify the correct files e.g.

```
remoteFiles:$RemoteFilePattern:[
  searchPatternRemote:''^.*\.dat.gz$'
  searchPatternRemoteExclude:'hum.*\.dat.gz^$'
  fromNamePattern:'.gz' toNamePattern:''
]
```

The above example will match all files ending in `.dat.gz`, but remove from this list any that start with 'hum'.



Note: The current version of SRS Prisma requires by default that all files be in the directory specified by the Remote Host. This means that file patterns cannot contain directory separators etc. unlike previous versions of SRS Prisma, unless recursion is used (see Section 3.8.1.15, Using Recursion, page 77).

To allow for differences in file naming between remote and local files, one or more sets of regular expressions and replacements can be used. When using VisAd, these can be added by clicking on the **Add** button of the **Case translation** panel.

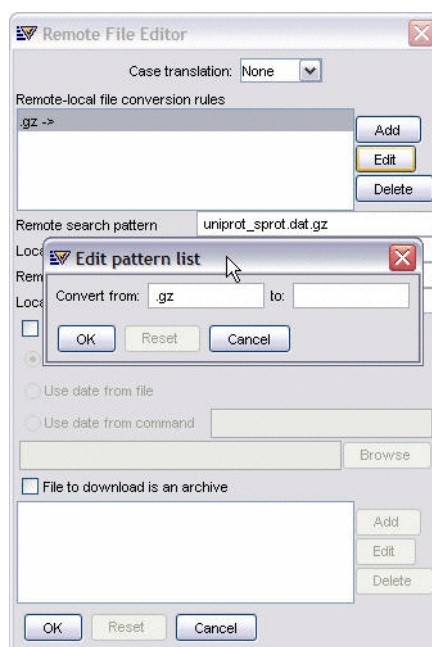


Figure 3.24 Adding a file conversion pattern.

To represent this directly within the `RemoteFilePattern` object, separate lists of regular expressions and their replacements are written to `fromNamePattern` and `toNamePattern`. Note that these lists *must* be of equal length.

Finally, case translation can be carried out by setting the Case translation option to **None**, **To upper** or **To lower** (reflecting the remote to local conversion). This can be set within the `RemoteFilePattern` object by setting the `caseTranslation` attribute to `none`, `upper` or `lower`.

3.8.1.11 Handling Archives

Prisma normally uses a one-to-one mapping of files on a remote site to those on the local site, but at times, it is useful to be able to download an archive. To signify that the file matched by the pattern on the remote site is an archive, check **File to download is an archive** and then click **Add** to add files that the archive contains.

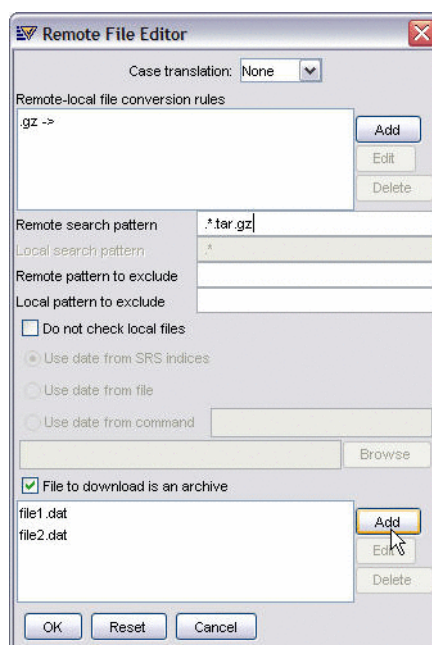


Figure 3.25 Specifying an archive.

In this case, the local search name pattern is ignored.

To achieve this using a `RemoteFilePattern` object directly, simply populate the `archiveContains` attribute with a list of files:

```
archiveContains:{"one.dat" "two.dat" "three.dat"}
```

Where the contents of the archive are not known, the best approach is to download the file, unpack it and allow a dependent database to deal with the files locally.

3.8.1.12 Alternative File Comparison Techniques

Normally, Prisma will carry out a comparison of remote and local files based on a comparison of remote vs. local timestamps. However, this is not always useful or appropriate. To ignore all local files and simply compare the remote files against a specified local date, check **Do not check local files** and select the appropriate option:

- Use date from SRS indices – take the timestamp from the SRS ID indices

- Use date from file – take the timestamp from the file specified
- Use date from command - take the UNIX time returned by the specified command

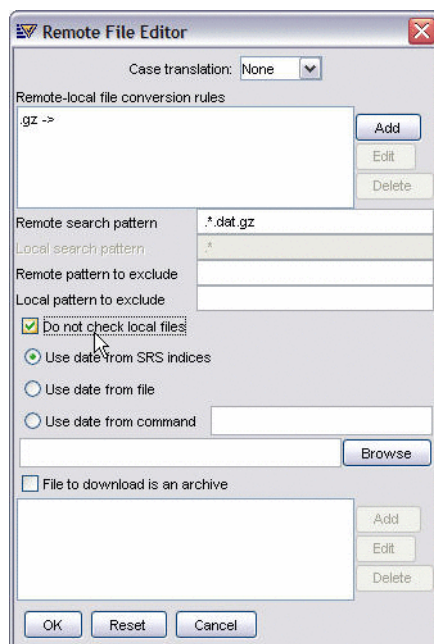


Figure 3.26 Specifying an alternative file comparison method.

To set this behavior directly using the `RemoteFilePattern` object, set `checkLocalDateOnly` to `y`. By default, this will use the date from the indices. To use the date from a file, specify the file as the attribute `localDateFile`, and to use the date from a command, specify the command as the attribute `localDateCommand`.

3.8.1.13 Specifying Preprocessing Commands

Data is often provided in the form of a tar and/or a zip file. The command to be used to extract such data can be specified by specifying a pre-processing (or 'unpack') command e.g. `gunzip uniprot_sprot.dat.Z`.

The command to extract the files is entered as a separate target in the `makefile`. As such, it runs in its own standard `sh` shell, and can call any appropriate command or program that is found in its inherited environment. Multiple commands should be separated by semicolons (`;`). The output of the commands can be redirected to files. If this is the case, care is required, because if the standard error is redirected, then errors will not be reported by the monitoring program.

File name substitution can be specified within the command, and the following substitutions are recognized:

%s

This specifies the full filename and its path from root. The path should be that of the offline data directory.

%n

This is used to specify the filename minus any extension (i.e., all characters up to but excluding the last dot in the complete filename)

%e

This specifies the filename extension (not including the dot).

%d

This specifies the path for a file (i.e. the offline data directory, including the terminal forward slash).

%N

This is a space-separated string containing a list of all new files downloaded.

%U

This indicates a space-separated string containing a list of all the updated offline files used.

%R

This is a space-separated string containing a list of all old files removed.

If any of the filename substitutions (`%s`, `%n` or `%e`) is used within an `unpack` command, the `makefile` will contain one target for each file being downloaded. The following command uses `%s` and will be executed as an independent `makefile` target for each file downloaded:

```
"$gunzip %s.gz"
```

The use of `%s` not only allows parallelization of the `unpackCommand`, but is also a necessary condition if the `prismamakefile` is to gain maximum benefit from the parallel indexing methods that can be specified for building SRS indices. It is then possible to begin generating the SRS indices whilst still downloading the remaining datafiles for a single library. Parallelization of multiple versions of the `unpackCommand` is possible for each, but 'chaining' of the download, unpack and build targets is implemented only for the `files` and `filesSize` methods of indexing. The command will only be called for the newly downloaded files. Existing files in the online directory are assumed to be in the correct format for indexing, and are simply linked or copied into the offline directory (if necessary) before indexing begins.

The following command uses the `%d` substitution string only, and hence is executed once only for the entire download set:

```
"$gunzip %d*.gz"
```



Note: Where no substitution strings are used to provide additional information, the administrator must ensure that downloaded files are in the correct locations and have the correct name pattern for unpack command to be executed correctly.

To add an unpack command, click **Add** on the panel labelled **Commands to preprocess new data** and enter the command in the dialog box. The **Edit** menu can be used to insert substitution strings as described above (or they can be entered directly).

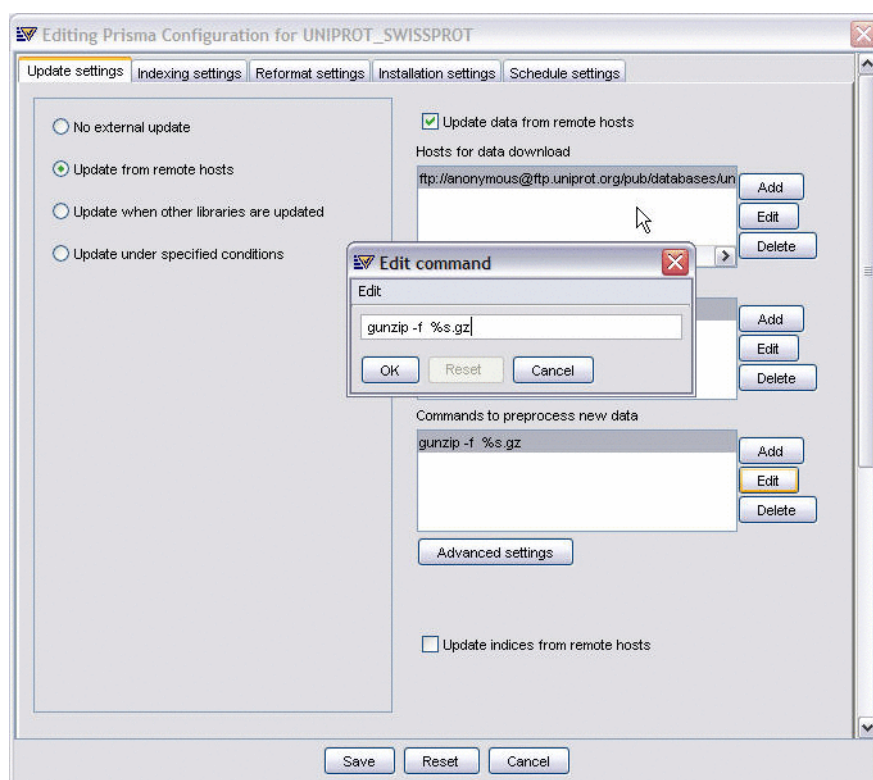


Figure 3.27 Adding an unpack command.

If editing the `Resource` object directly, the unpack command(s) can be specified as members of the `unpackCommand` list attribute e.g.:

```
unpackCommand:{
  "$gunzip %s.gz"
}
```



Note: Although a single `unpackCommand` is generally used, it is possible to specify multiple commands that are executed in order. This arrangement has the advantage of being able to specify a command that is executed on a per-file basis (using `%s` in this case), and a second command that is executed once only. During the execution phase, the first command is executed for each file, then when all of these have finished, the second command is executed once.



Note: Use of mixed per-file and per-database `unpackCommand` strings in a list will break the parallel chain of download-pretrans-build commands that SRS Prisma normally creates, as the per-database command introduces a bottle-neck.

3.8.1.14 Advanced Options

A number of advanced options are provided to increase the flexibility of SRS Prisma. These can all be accessed by checking the **Advanced settings** box or by setting the appropriate `Resource` attribute.

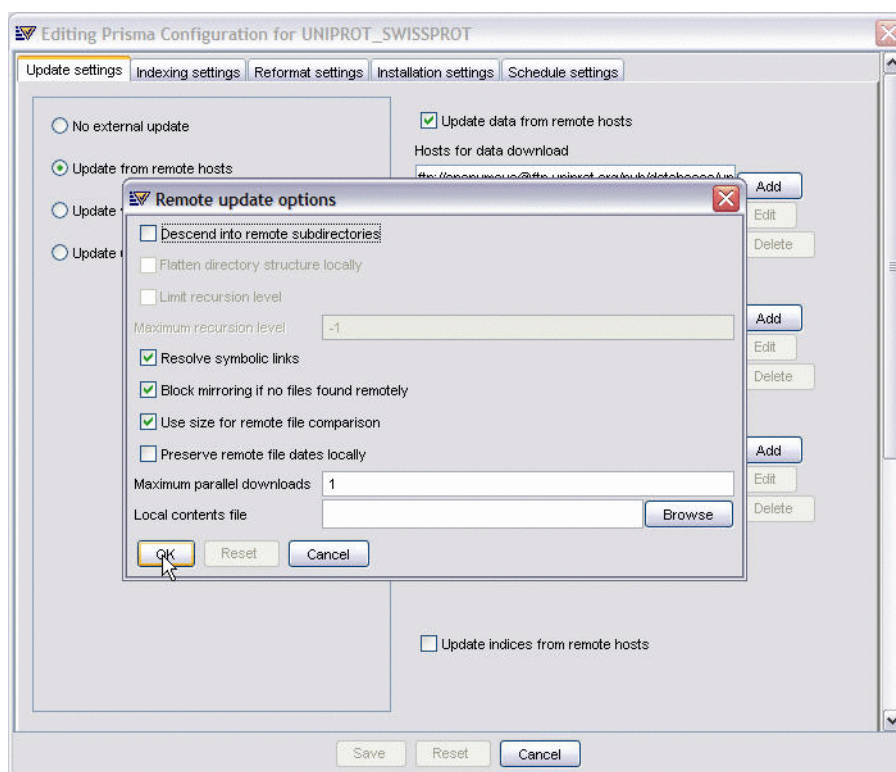


Figure 3.28 Specifying advanced options.

3.8.1.15 Using Recursion

Normally, SRS Prisma assumes that all remote files are on a single directory. However, sometimes it is necessary to use files from multiple directories. To allow this, SRS Prisma can recursively search remote and local sub-directories, and download new files to new local directories created as required. To enable this, check **Descend into remote subdirectories** or set the `Resource` attribute `subDirectories` to `yes`.

Alternatively, files in sub-directories on remote sites can be saved to the same level locally, thus flattening the directory structure. To activate flattening, check **Flatten directory structure** or set the `Resource` attribute `subDirectories` to `flatten`.

Note that this may cause problems with multiple identically-named files in different sub-directories. An alternative strategy is to not use the automatic flattening mechanism, but use file name patterns to create a local file name by replacing '/' in the remote file path with '_' or similar.

By default, SRS Prisma will continue to descend through the structure until all terminal directories are found. To control this, the maximum level of recursion can be set using the **Maximum recursion level** entry box, or setting the `maximumRecursion` attribute of the `Resource` object.

The remote and local search patterns and file patterns can be used normally, although note that they can in fact apply to any of the relative path, allowing directory names to be used to discriminate between files.

3.8.1.16 Symbolic Links

By default (and where the server allows), symbolic links on a remote site will be resolved and the original file. To change this behavior so the links are preserved locally, uncheck the **Resolve symbolic links** checkbox or set the `followLinks` attribute of the `Resource` object to `n`.

3.8.1.17 Sanity Checking

SRS Prisma not only notes new or updated files on a remote site, but also earmarks, for deletion, any local files that are no longer present on the remote site. However, if reorganization occurs on the remote site, and the specified download directory no

longer contains any matching files, SRS Prisma has the potential to delete all local flat-files for that library. To minimize the chances of this happening, SRS Prisma will abort remote checking for a library if it finds no matching files on the remote site. However, this is occasionally the correct state of affairs. For example, for update libraries which contain no files immediately after a release of the main parent library. In such cases, the checkbox **Block mirroring if no files found remotely** should be unset, or the `Resource` attribute `checkNoFiles` can be set to `no` to override this default behavior.

3.8.1.18 Remote File Size Comparison

By default, SRS Prisma takes both file size and date into consideration when determining which remote files should be downloaded. For a remote file to be downloaded, it should be both newer and smaller than the local file. This is to ensure that partial downloads will not be treated as complete local files. However, this is not always ideal behavior, so SRS Prisma can be forced to use file dates only for comparison. This behavior can be specified on a per database basis by unchecking the **Use size for remote file comparison** checkbox, or setting the `checkFileSizes` attribute of the `$Resource` class object to `yes`.

3.8.1.19 Maximum Parallel Downloads

If the files to be downloaded form part of a library that is to be indexed in SRS using either the `files` or `fileSize` parallel mode, then a separate connection will be formed for each file. As a result, if large numbers of these were allowed to operate in parallel, they could exceed the bandwidth of the site. To avoid this, the targets are 'chained' in their dependencies, so that the number of simultaneous connections for any given databank can be limited on a per-library basis.

The number of chains can be specified using the **Maximum parallel downloads** option, or by setting the `mirrorChains` attribute of the `Resource` object.

Controlling the number of chains cannot limit the total number of simultaneous connections where multiple databanks are to be updated. Instead, see Section 3.2.2.1, General Settings, page 27.

3.8.1.20 Local Contents File

Normally, SRS Prisma lists files locally in the online data directory using `ls -l` to find names, dates etc. Alternatively, the name of a file containing a listing of the local directory can be specified. This will be used instead of listing the online directory, and should be in the same format as the output of `ls -l`. Note that the offline directory will be listed as normal.

3.8.1.21 Testing Remote Settings

Although the most complete way to test a remote update setting is to run SRS Prisma on the selected library, this may not always be the quickest way of checking the settings. An alternative is to create a sub-directory in `$SRSFLAGS` with the same name as your library, and run:

```
% $SRSLION/prisma/dbSync.i -libName <libraryName> -dirName <onlineDataDirectory> -  
offDirName <offlineDataDirectory> -flagsDir $SRSFLAGS -checkonly
```

The output should help you find out if the connection is working and finding the files you want.

3.8.2 Remote Index Updates

In addition to downloading new data files from remote hosts, it is also possible to download pre-built SRS indices from a remote host (where available). This is carried out on a per-library basis, and allows all necessary indices and OM files to be downloaded. All required links and virtual libraries are then built locally.

This provides the possibility to much reduce indexing time, or to mirror SRS libraries between installations on different sites. The next section describes how this can be achieved.



Information: SRS indices must be used for querying/retrieval with identical data to that used for indexing, or they will be unusable. In addition, care should be taken to

ensure that the same configuration (parsers and Icarus configuration files) or querying may not work properly.

3.8.2.1 Specifying Remote Hosts

As with data, at least one remote host is needed from which indices are downloaded. To add a new host using VisAd, ensure that **Update indices from remote hosts** is selected, and click **Add** on the **Remote index hosts** panel. The dialog presented is identical to that used to specify hosts for data update (see Section 3.8.1.3, Specifying Remote Locations, page 56):

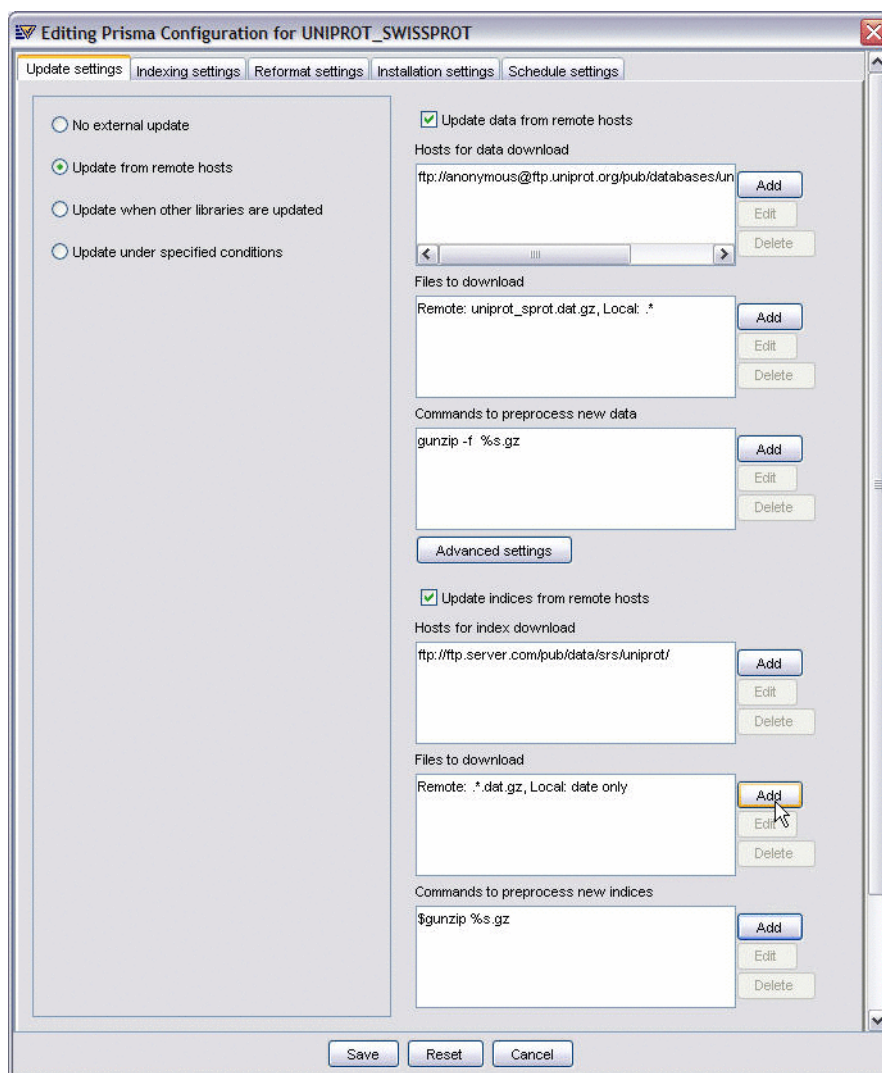


Figure 3.29 Specifying a remote index host.

To add a remote host for indices directly to the `Resource` object, add a new instance of a `RemoteHost` to the `remoteIndexHosts` list attribute:

```
remoteIndexHosts: {
  $RemoteHost: [url:"ftp://ftp.server.com/pub/data/srs/uniprot"]
}
```

3.8.2.2 Specifying Remote Index Files

As with data, at least one remote file pattern is needed, but these are configured differently. To add a new pattern, click on the **Add** button on the panel labelled **Files to download**:

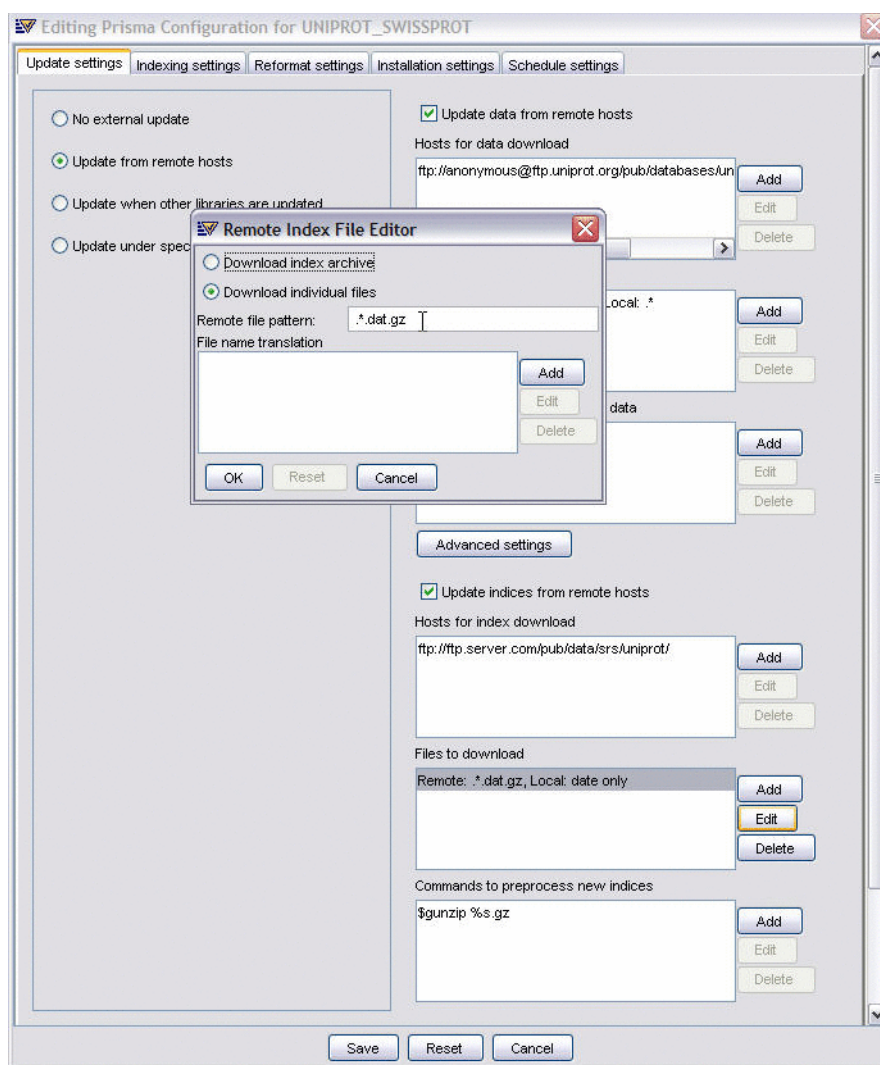


Figure 3.30 Specifying a remote index file pattern.

If editing the `Resource` object directly, new `RemoteFilePattern` objects are added to the `remoteIndexFiles` list.

The specification for remote index file patterns can be of two kinds, index archive, or individual index files.



Note: If **Download index archive** is selected, the file matched by **Remote file pattern** is assumed to be a single archive file containing all necessary indices and OMs needed for the library.



Note: If **Download individual files** is selected, then the remote file pattern should match all indices on the remote site. Note that with this setting, a single file pattern of this type will automatically be expanded to look for and download all required indices. Remote file pattern conversion rules can be used to indicate where file names contain differences like `.gz`.

If configuring directly, then to specify an index archive, the following form of `RemoteFilePattern` object is used:

```
$RemoteFilePattern: [
  searchPatternRemote: "enzymeIndices.tar.gz"
  checkLocalDateOnly: y
  archiveContainsIndices: y
]
```

To specify that individual files should be downloaded, then use:

```
$RemoteFilePattern: [
  searchPatternRemote: "enzyme.*.gz"
  fromNamePattern: { ".gz" }
  toNamePattern: { "" }
  checkIndexFiles: y
]
```

3.8.2.3 Specifying Preprocessing Commands

If the indices require unpacking (unzipping, or extraction from a tarball), then one or more unpack commands may be specified. This is carried out in exactly the same way as for data:

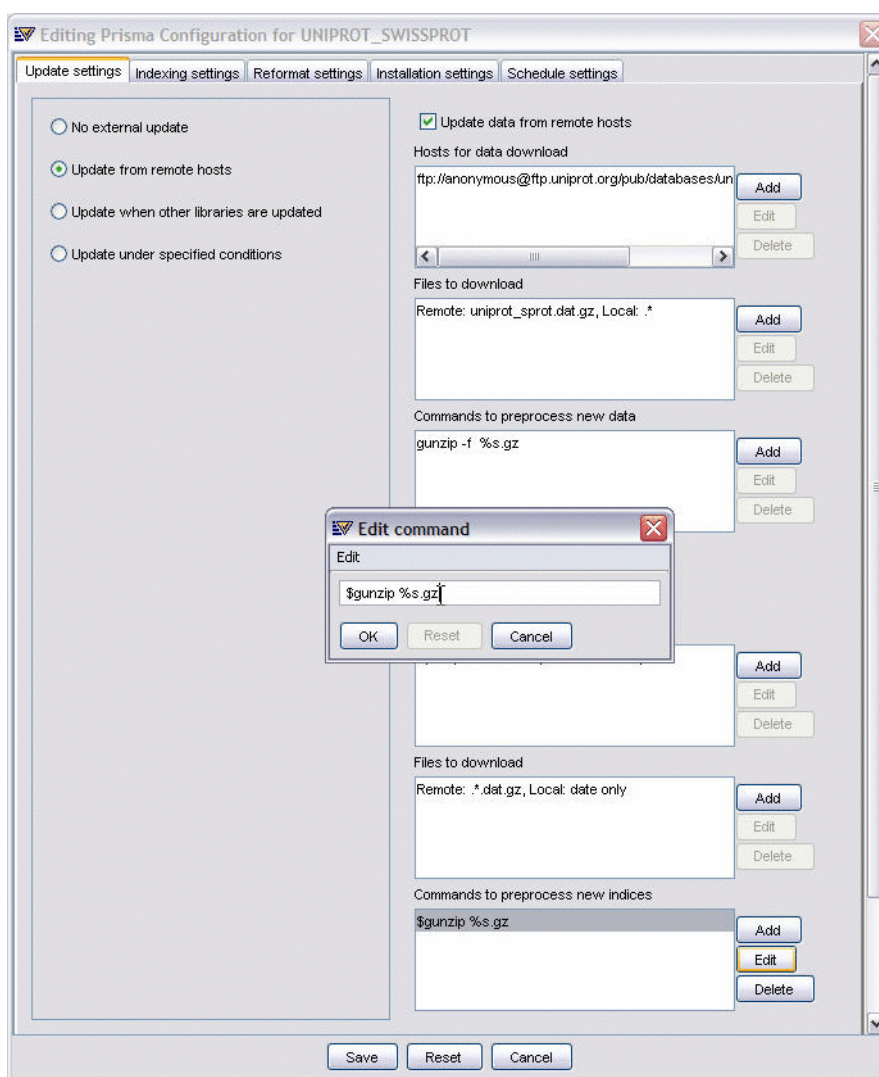


Figure 3.31 Specifying index preprocessing commands.

For direct configuration with the `Resource` object, one more commands can be entered into the `unpackIndexCommand` attribute.



Note: When dealing the indices, the online index directory is treated as the online data directory, and the offline index directory is treated as the offline data directory.

3.9 Configuring the Indexing Phase

VisAd allows some aspects of the indexing process to be controlled using the **Index settings** tab:

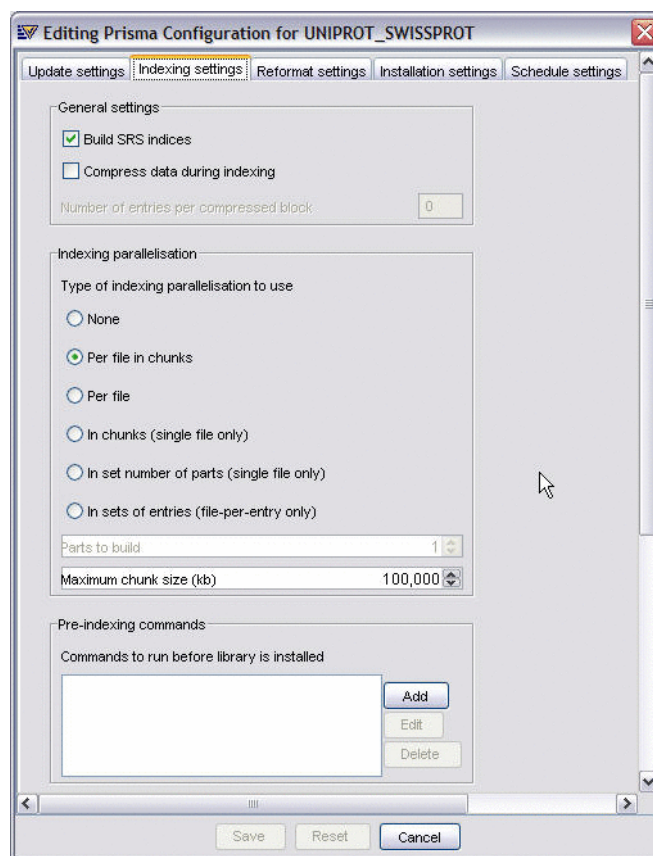


Figure 3.32 The Index Settings tab.

3.9.1 Indexing Settings

The indexing phase of the Prisma update can also be configured. This is done using the **Indexing settings** tab. This can be used to set the compression level (see Chapter 14, Data Compression, page 223 of the *SRS Advanced Administrators Guide*) and the parallel type and associated settings can be set here. For more information about which parallel types are most appropriate, see Section 10.4, Parallel Indexing, page 329 of the *SRS Basic Administrators Guide*.

3.9.2 Non-SRS Libraries

Not all libraries updated by SRS Prisma need to have valid, indexable data from which SRS indices are produced. It can be very useful to have a non-SRS library to download and process new files, or to perform post-processing commands as a dependent library (see Section 3.14, Dependent Libraries, page 119).

To specify a library as non-SRS, uncheck the **Build SRS indices** check box in the **Indexing settings** tab, or set the `Resource` attribute `noSRS` to `y`.

3.9.3 Pre-indexing Commands

Commands can be specified that are run prior to indexing, to allow commands such as partial index cleanup commands to be run. Note that these are not dependent on download or local copy commands, and could be run at any point before indexing starts. To add a pre-indexing command, click **Add** on the **Pre-indexing Commands** panel:

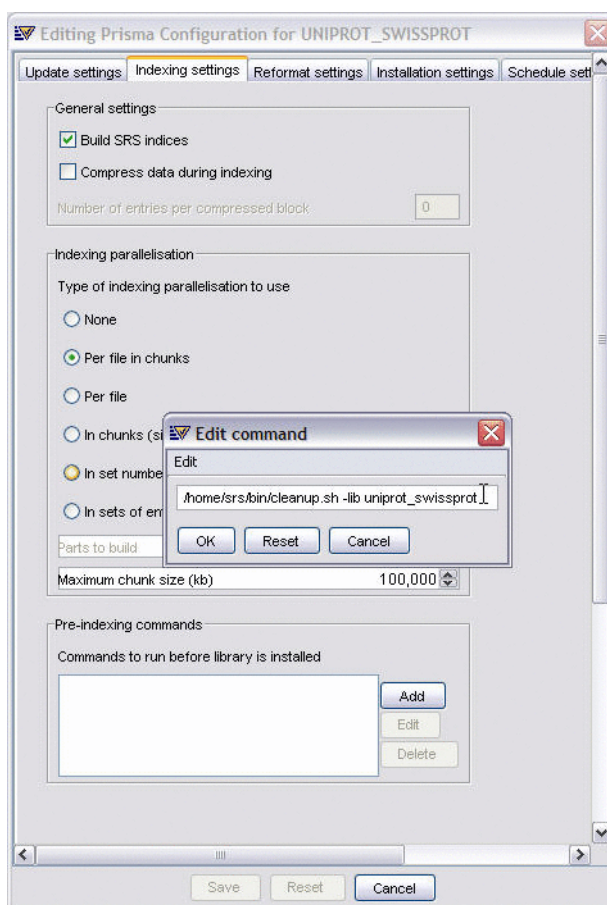


Figure 3.33 Adding pre-indexing commands.

Alternatively, the commands may be set directly in the `Resource` object using the `preIndexCommands` attribute, which accepts a list of strings.

3.10 Configuring Postprocessing

After SRS Prisma has completed indexing, it is possible to run additional actions. This can include automatically generating FASTA and BLAST files from the data set for the library in question, or from a specified query, or can include the execution of specified shell commands on a per-library or per-file basis. The following sections

outline how to configure SRS Prisma to carry out these commands. The **Reformat settings** tab can be used to configure post-processing ('reformat') commands:

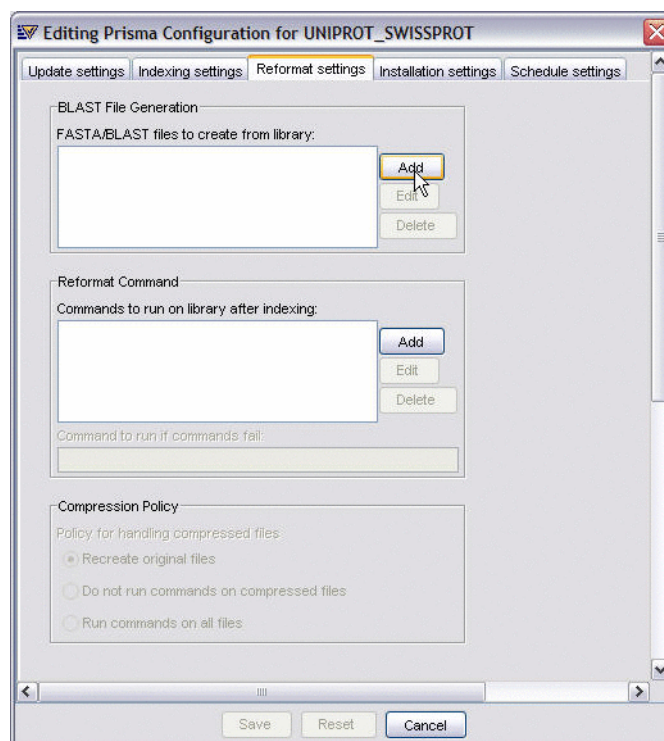


Figure 3.34 The Reformat Settings tab.



Important: If any command (including query-based FASTA generation) depends on another library, or links between libraries, then to ensure consistency, the commands *must* be associated with a dependent library that is updated after any libraries involved have been updated. See Section 3.14, Dependent Libraries, page 119 for more details on dependent libraries.

3.10.1 Generating FASTA and BLAST Files from Sequence Data

SRS Prisma is capable of automatically creating FASTA files from sequence data that it updates, and of generating BLAST indices for those files.

To add generation a FASTA/BLAST file to a library, open the library for editing using VisAd, open the **Reformat settings** tab, and click **Add** on the panel labelled **Blast File Generation**. To add this to the `Resource` object, directly, you need to add a new instance of a `PrismaBlastFile` object to the `blastFiles` list attribute of the `Resource` object:

```
blastFiles:{
  $BlastFile:[
    dbName:uniprot_swissprot type:file
    fastaConverter:"sp2fasta %I > %O"]
  }
```



Note: For every BLAST file added, a unique name must be specified that is used as the name of the FASTA file and the basename of the BLAST indices.

The settings used for different types of FASTA generation and BLAST indexing are discussed in the next sections.

3.10.1.1 Query-Based FASTA Generation

Query-based FASTA generation converts entries from a specified SRS query into a FASTA file. This uses a supplied Icarus script to convert entries from EMBL, SWISSPROT or GENBANK format into a simple form of FASTA (headers are of the form `>database:id description`). This is carried out in multiple parallel processes to generate partial FASTA files that are concatenated together once finished.

This mechanism is extremely powerful in allowing any SRS query (including links) to be specified, but it should be noted that it is slower than calling an external utility to parse entire data files, and is limited in the output and input formats.

The following dialog shows how to configure a FASTA file to be generated from an SRS query:

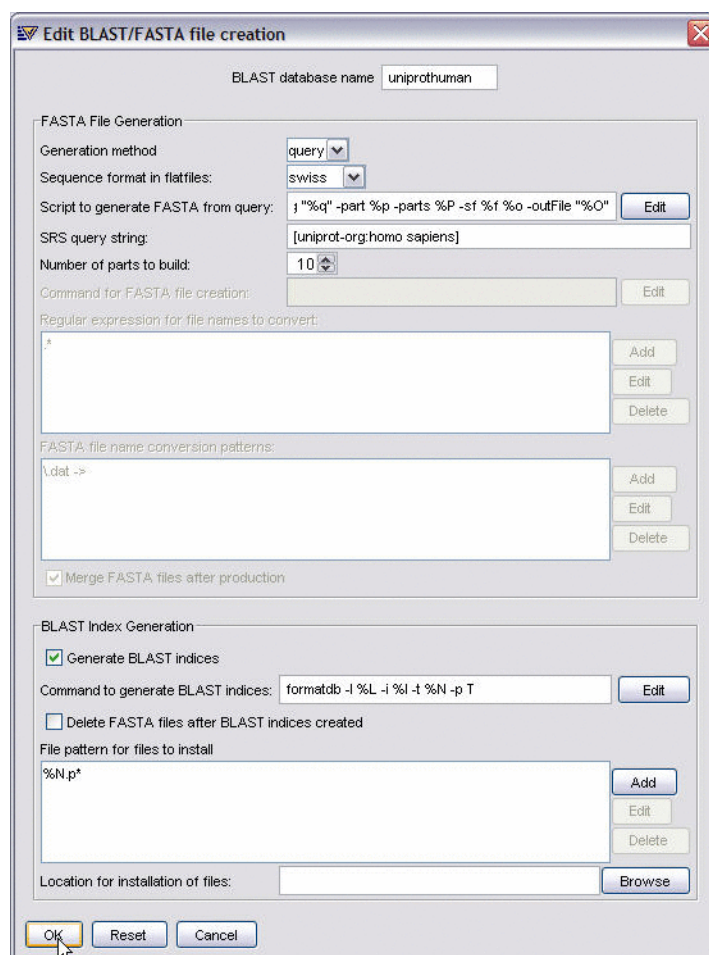


Figure 3.35 A query-based FASTA file.

The following values can be set using the dialog (BlastFile attributes in brackets):

Database name (dbName)

This is the title of the database generated, and the name of the file created by merging partial FASTA files.

Generation method (type)

This should be set to 'query' to indicate query-based conversion.

SRS query string (`queryStr`)

This should contain an SRS query to return entries containing the desired sequences.

Sequence format in flatfiles (`sequenceFormat`)

This is the format of the sequence in the entries retrieved by SRS. This should be set to `embl`, `swiss` or `genbank`.

Script to generate FASTA from query (`fastaConverter`)

This is the path of the script. An alternative script can be supplied if required (see note below).

Number of parts to build (`numParts`)

This is the number of processes to run in parallel to generate the FASTA file. Each process generates FASTA from a portion of the entries retrieved, and the partial files are concatenated at the end.

The equivalent `BlastFile` object is:

```
$BlastFile: [
  dbName:"uniprothuman"
  type:query
  queryStr:"[uniprot-org:homo sapiens]"
  partN:10
]
```



Note: By default, the sequence data in the libraries queried **MUST** be one of the specified formats. It is possible to write and specify for use a replacement script that parses any format data and writes any format of FASTA, but the details of such a script are beyond the limits of this guide. Please contact SRS support for more advice on this topic.

3.10.1.2 Files-Based FASTA Generation

Alternatively, FASTA files can be generated directly from the data files associated with a library (or a subset thereof) using a third-party utility such as `sp2fasta`. These FASTA files can then be merged to a single file which be left unused, or left as individual files.

The following dialog shows how to configure a BLAST file to be generated from sequence data files:

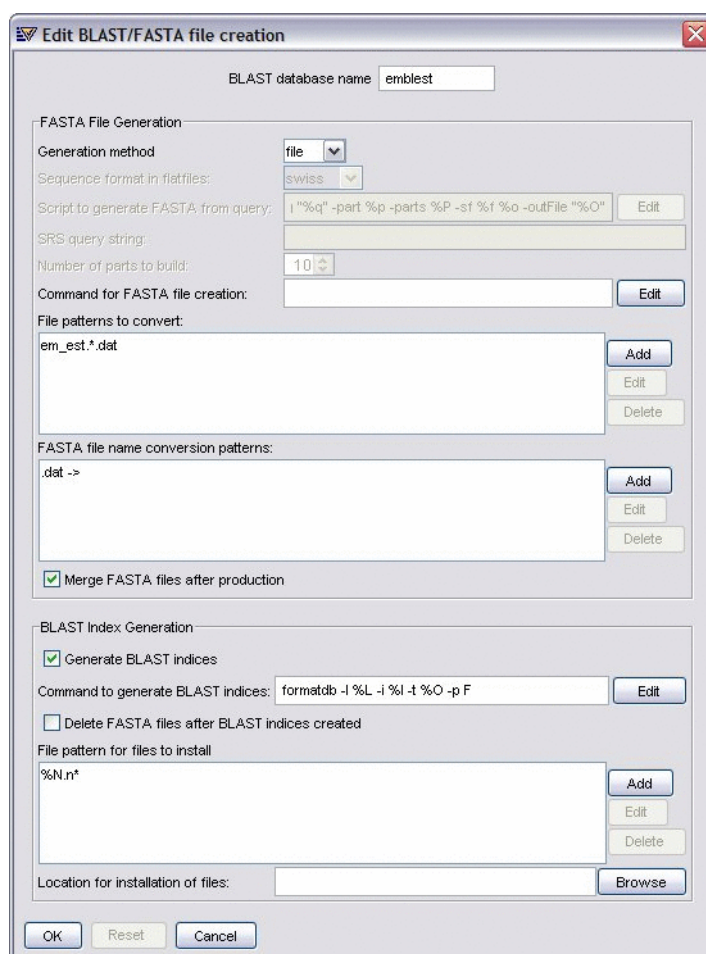


Figure 3.36 Configuring BLAST indices generation.

The following values can be set using the dialog (BlastFile attributes in brackets) although suggested values are inserted automatically where possible:

Database name (dbName)

This is the title of the database generated, and the name of the file created by merging partial FASTA files.

Generation method (type)

This should be set to 'file' to indicate per-file conversion.

FASTA converter (fastaConverter)

This must be set to the command used. This must include the following placeholders:

- %I – replaced by path to input file
- %O – replaced by path to output file
- %N - database name

File patterns to convert (filePatterns)

This is a list of regular expressions matching the data files that should be converted. If left empty, the default is used is '.*'.

FASTA file name conversion patterns (fromNamePattern and toNamePattern)

This is a list of regular expressions to convert the original file name into the name of the FASTA file. For instance, to convert a file called `em_est1.dat` to `em_est1`, the patterns should be `.dat` and `""`.

Merge FASTA files after production (mergeFiles)

This controls whether to concatenated individual FASTA files into a single file, or leave them as individual files. If a single file only is produced, it is renamed.

The equivalent `BlastFile` object is:

```
$BlastFile: [
  dbName: "embltest"
  type: file
  fastaConverter: "sp2fasta %I > %O"
  mergeFiles: y
  filePatterns: { "em_est.*.dat" }
  fromNamePattern: { "\\..dat" }
  toNamePattern: { "" }
]
```



Information: SRS Prisma does not include the utilities necessary for converting data files into FASTA. These should be obtained and installed separately.

3.10.1.3 Use of Existing FASTA Files

Finally, if the files used by an SRS library are already in FASTA format, it is possible to produce a concatenated FASTA file and generate BLAST indices from them. The following dialog illustrates a configuration to produce a single FASTA file from multiple FASTA files associated with a library:

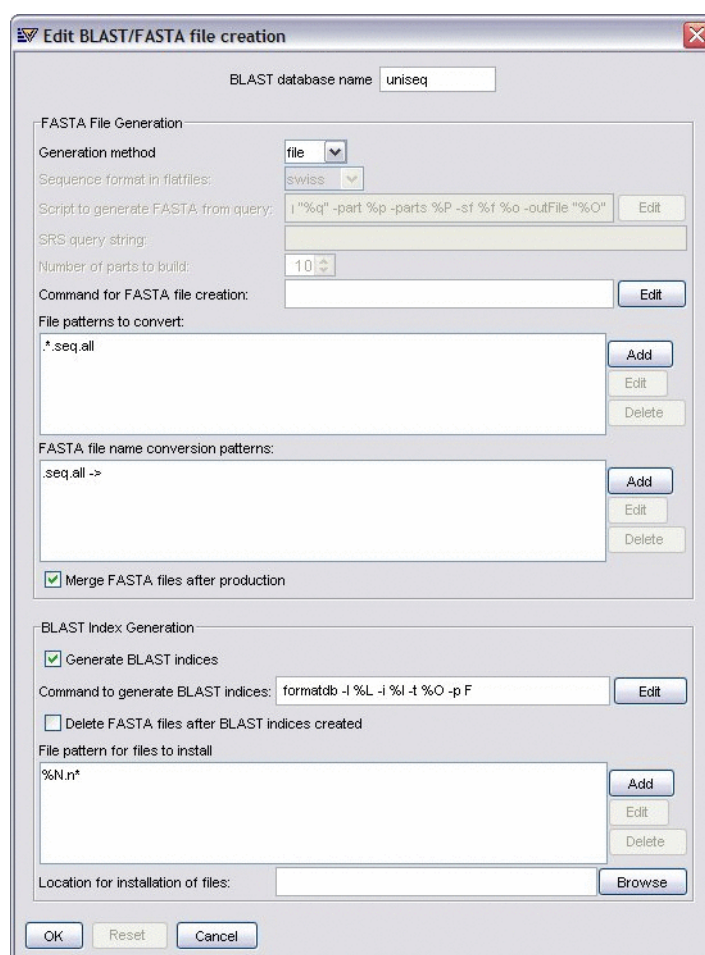


Figure 3.37 Use of existing FASTA flat-files.

The following values can be set using the dialog (BlastFile attributes in brackets):

Database name (dbName)

This is the title of the database generated, and the name of the file created by merging partial FASTA files.

type (type)

This should be set to 'none' to indicate no conversion is needed.

File patterns to convert (filePatterns)

A list of regular expressions matching the data files that should be converted. If left empty, the default is used is '.*'.

FASTA file name conversion patterns (fromNamePattern and toNamePattern)

A list of regular expressions to convert the original file name into the name of the FASTA file.

Merge FASTA files after production (mergeFiles)

Whether to concatenate individual FASTA files into a single file, or leave them as individual files. If a single file only is produced, the original file is copied.

The equivalent `BlastFile` object is:

```
$BlastFile:[
  dbName:"uniseq"
  type:none
  mergeFiles:y
  filePatterns:{"*.seq.all"}
  fromNamePattern:{"\\..seq.all"}
  toNamePattern:{""}
]
```

Note that it is also possible to uncheck **Merge files** and generate BLAST indices for each FASTA file individually.

3.10.1.4 BLAST File Generation

Once FASTA files have been generated, the next step is to generate BLAST indices for them. The following dialog illustrates how this is configured:

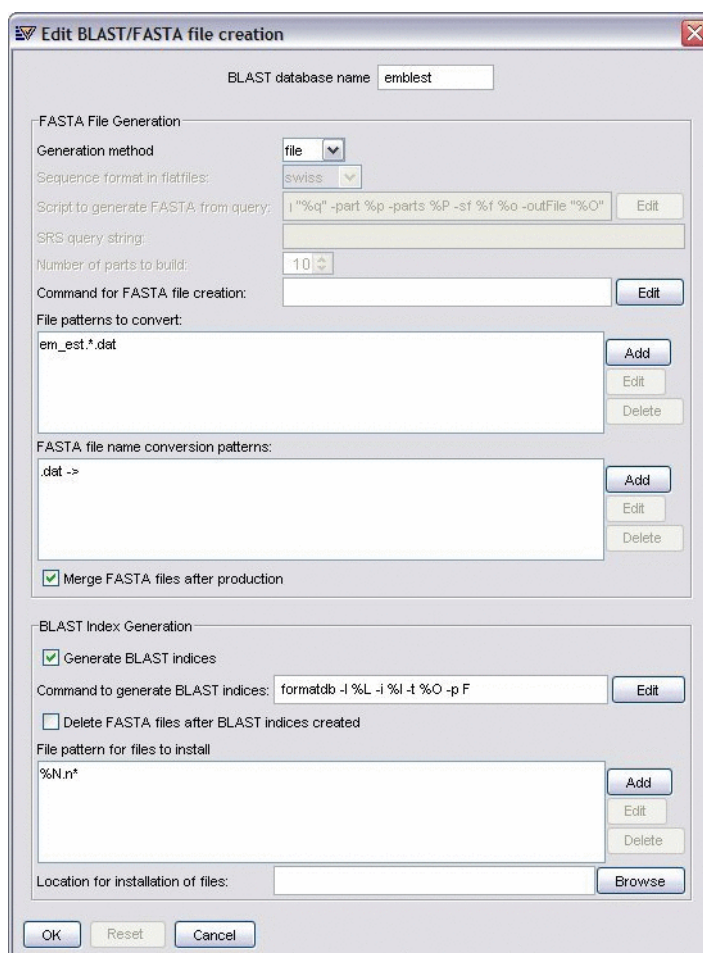


Figure 3.38 Generating BLAST files

The following values can be set using the dialog (BlastFile attributes in brackets):

Generate BLAST indices (createBlastFiles)

If this is checked, BLAST indices will be generated using the settings below.

Command to generate BLAST indices (blastConverter)

This is the command used to generate the BLAST indices. This accepts the following place-holders:

- %L - logfile
- %I - input file
- %O - output file title



Note: SRS Prisma does not include the utilities needed to generate BLAST indices from FASTA files (e.g. `formatdb`). These should be obtained and installed separately.

Delete FASTA files after BLAST indices produced (`keepFasta`)

If the FASTA files are not required after BLAST indices have been produced, SRS Prisma can delete them if this box is checked. By default, the FASTA files will be kept.

File pattern for files to install (`blastFilePatterns`)

This is a list of regular expressions matching the final BLAST files to install online. This accepts the placeholder `%0` to be replaced the output file title of the BLAST files.

Location for installation of files (`installLocation`)

This is the location where the BLAST and FASTA files will be installed during the move phase. If not set, the `$BdbRoot` variable used by SRS tools will be used.

The equivalent `BlastFile` object is:

```
$BlastFile:[
  dbName:"emblest"
  type:file
  fastaConverter:"sp2fasta %I > %0"
  mergeFiles:y
  filePatterns:{"em_est.*.dat"}
  fromNamePattern:{"\\.dat"}
  toNamePattern:{""}
  blastConverter:"formatdb -l %L -i %I -t %0 -p F"
  keepFasta:y
]
```

3.10.2 Running Reformat Commands

In addition to FASTA/BLAST file generation, one or more additional commands ('reformat commands') can be run to post-process or reformat data after indexing. The commands specified are called after the remote datafiles have been downloaded and unpacked. Commands are called once for each file in the dataset if filename substitutions (`%s`, `%n`, or `%e`) have been specified. Otherwise, it is called once for each

library, after any SRS indices have been built. It is the responsibility of the command itself to manage any extra datafiles it creates, where it stores them, or how it archives any older versions of these files. This management must be compatible with the install and archive options specified for each SRS library. If multiple commands are specified, they are called in order, using appropriate parallelization.

To add a reformat command, select the **Reformat settings** tab and click **Add** in the **Reformat command** panel. The command can then be edited.



Figure 3.39 Adding a reformat command.

The command can include the following placeholder strings:

%s

This specifies the full filename and its path from root. The path should be that of the offline data directory.

%n

This is used to specify the filename minus any extension (i.e., all characters up to but excluding the last dot in the complete filename).

%e

This specifies the filename extension (not including the dot).

%d

This specifies the path for a file (i.e. the offline data directory, including the terminal forward slash).

%N

This is a space-separated string containing a list of all new files downloaded.

%U

This indicates a space-separated string containing a list of all the updated offline files used.

%R

This is a space-separated string containing a list of all old files removed.

These can be entered directly, or using the **Edit** menu.

To specify reformat commands using the `Resource` object directly, specify the commands as a list of strings with the `reformatCommand` attribute:

```
reformatCommand:{  
  "reformatData.sh %s > %n"  
}
```

3.10.2.1 Postprocessing Failover Commands

In addition, a failover command may be specified that can be run if one or more post-processing command fails. This allows processes started by the pre-processing commands to be stopped cleanly and any clean-up commands to be executed. A failover command may be entered in the **Failover command** box on the **Reformat Settings** panel, or set directly using the `Resource reformatFailureCommand` attribute.

3.10.2.2 Postprocessing and Compressed Files

When using SRS compression (see Data Compression, page 223 of the *SRS Advanced Administrators Guide* for more details), post-processing commands which are run on a per-file basis may encounter a mixture of compressed and uncompressed files. The behavior of SRS Prisma at this point may be controlled using the **Compression policy** panel of the **Reformat settings** tab:

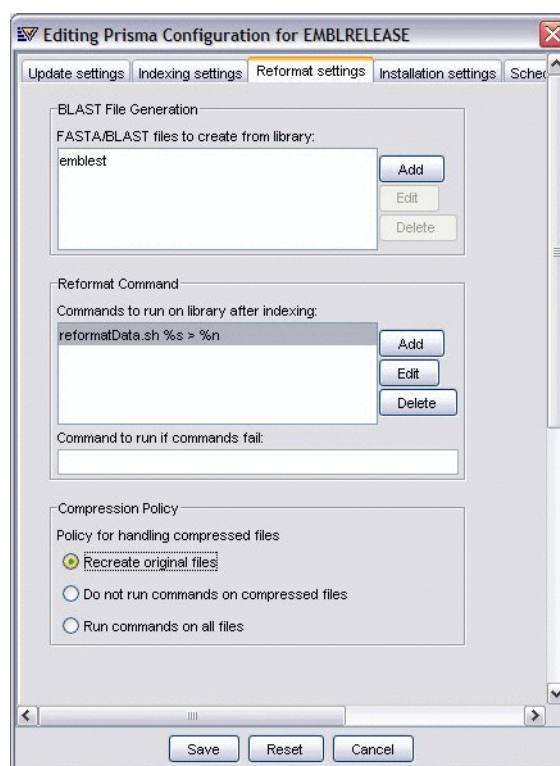


Figure 3.40 Setting compression policy.

or by setting the Resource attribute `reformatCompressedFiles` to the appropriate option. The following options are available:

Recreate original files (`recreate`)

Any compressed files are transiently uncompressed and concatenated to reproduce the original files. These files are deleted as soon as no longer required.

Do not run commands on compressed files (skip)

Per-file commands are only executed on uncompressed files

Run commands on all files (none)

Per-file commands will be run on compressed and uncompressed files alike, and are expected to behave appropriately.

3.11 Installation Settings

After all required updating steps have been carried out, SRS Prisma will install the data and/or indices associated with a successful library online. This phase can be controlled for each library to allow the appropriate commands to be run.

3.11.1 Installation Mechanisms

SRS Prisma supports four different mechanisms for installing data and indices online. These can be chosen using the **Installation settings** tab:

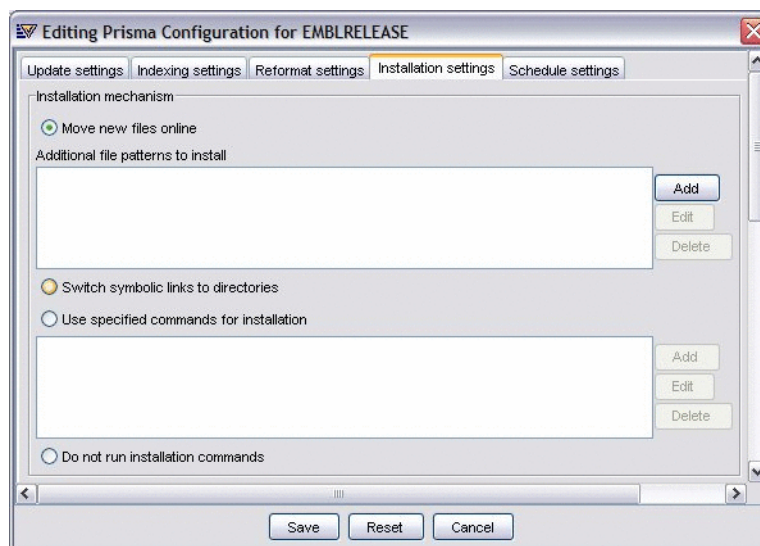


Figure 3.41 The Installation settings tab.

These can be set in the `Resource` object directly by setting the `installationType` attribute to the appropriate option (shown in brackets).

3.11.1.1 Move New Files Online (*move*)

This is the default option and will move any new or modified data files from the offline data directory to the online data directory. Any SRS index files will also be moved from the offline index directory to the online index directory. It does not move any additional files which might have been generated during the reformat phase (apart from those BLAST/FASTA files automatically generated by SRS Prisma). A list of UNIX file patterns (not regular expression) should be specified for installation separately using the **Additional files to install** panel:

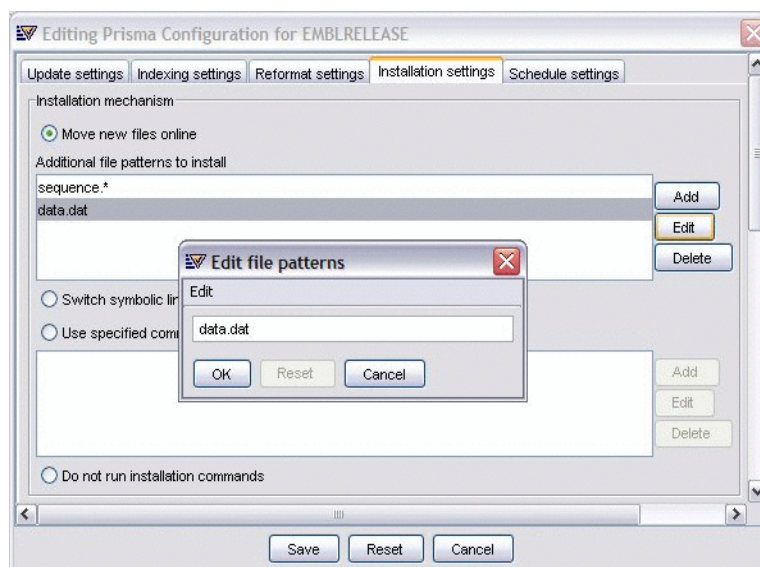


Figure 3.42 Adding extra files to install.

By default, a single move command is used for each of pattern specified to move the pattern from the offline directory to the online directory. However, the following placeholders can be also used to generate multiple move commands (one for each file where used):

- `%s` – replace with full name of each file (not including directory).
- `%n` – replace with basename of each file.

- %e – replace with extension of each file.

File patterns can be specified directly using the `installFiles` attribute of the `Resource` object:

```
installFiles: { "sequence.*" "data.dat" }
```



Note: The specified files must exist after any successful update, local or remote. The use of this attribute is not required for FASTA/BLAST files generated automatically by SRS Prisma.

3.11.1.2 Switch Symbolic Links (*switchlink*)

This option requires that the `dir` and `offDir` attributes for the relevant library in the `srsdb.i` file be set as symbolic links, each one pointing to a physical directory. These links must initially be set manually. SRS Prisma will install the downloaded datafiles by switching the symbolic links for `dir` and `offDir`.

SRS Prisma will not attempt to install new index files for `switchlink` databases. Instead, the online and offline index directories for a `switchlink` database should be subdirectories of the data directory, and as such they will be implicitly installed when the data directories are moved.

3.11.1.3 Use Specified Commands for Installation (*command*)

This option calls additional shell commands to carry out installation.

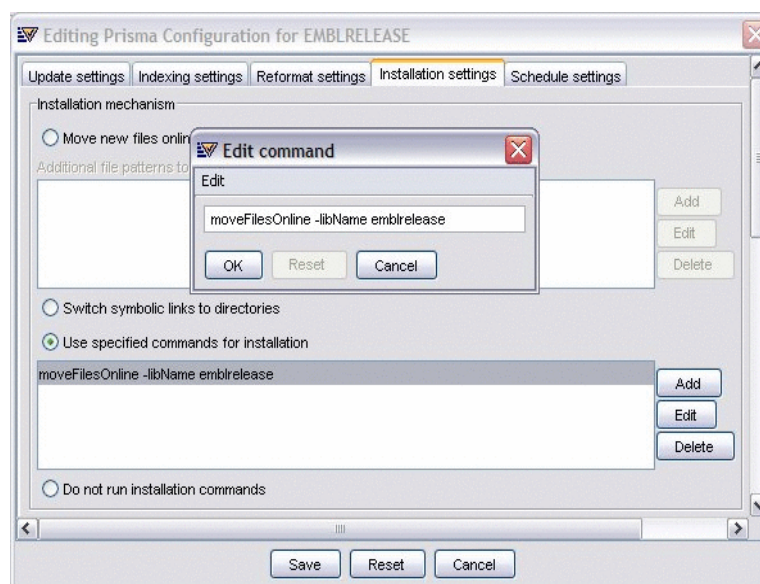


Figure 3.43 Specifying installation commands.

These commands do not allow filename substitution, although %d, %N, %R and %U are available and are called once for each library.

If this option is selected, the command must handle all the necessary processing for installing any files in the offline data directory to their final location, and take care of any archiving of data files that might be overwritten.

The offline data directory should be left completely empty, ready for the next time SRS Prisma is called. SRS Prisma will still handle moving and archiving of any SRS index files that are generated for this databank.

3.11.1.4 Do Not Carry Out Installation (*none*)

This option will leave any datafiles or index files in their respective offline directories (unless they were moved as part of a `reformatCommand`). The operator must then move these files online manually.

3.11.2 Archiving

By default, SRS Prisma will delete old flat-file and index sets when it installs new data online. However, it is often prudent to ensure that old flat-file and index sets are archived, so that, in the event of the new data set not being as expected (changes at the remote site, file corruption, and so on), it is relatively easy to restore the original files for users to access.



Note: Archiving is only available with the default mechanism for installation. Switchlinking is not compatible with archiving.

The available archiving options can be set using the 'Archive'

3.11.2.1 Do Not Archive Old Data and Index Files (*delete*)

This is the default option. The `delete` option does not attempt to archive files, and will delete any local datafiles it finds that are no longer present at the remote site, or any online files that have been replaced. This could cause problems if, for example, the directory where the data is stored at the remote site changes, because SRS Prisma might be fooled into deleting *all* the local data files for this library. As a result it is strongly recommended that one of the other options be selected, or the `checkNoFiles` option be set (see Section 3.8.1.17, Sanity Checking, page 77).

3.11.2.2 Archive by Adding Extension to Filename (*rename*)

If set, the specified file extension is appended to the filename for each modified or deleted data file, and each index file, but they remain in their respective online directories. This is a single stage archive, a second update to the database may overwrite saved files. In each case, if a problem occurs with a database during an update, it will be necessary to move back or rename the saved files, manually, to restore the previous data set and SRS indexes.

The extension is specified directly in the `Resource` object using the `removeExt` attribute.

3.11.2.3 Archive by Moving to Another Location (*move*)

If set, any changed or deleted data files are first moved into the specified directory. All the old index files are also moved into this directory. This is a single stage archive - if the database is updated again, any files already in the specified directory may be overwritten. If the databank stores datafiles in subdirectories, then the subdirectory tree structure will be recreated:

The archive directory is specified directly in the `Resource` object using the `removeDir` attribute.



Note: Old archive files are simply overwritten and never specifically deleted by SRS Prisma. For this reason, a little care is required so as not to reintroduce historically deleted datafiles during this procedure.

3.11.2.4 Archiving and Online-Offline Data Directories

Normally, SRS Prisma will download flat-files for each library into a separate offline data directory and build any required SRS indices in the offline index directory. If you want the datafiles to be downloaded directly into the online directory, this can be specified by setting the `dir` and `offDir` attributes, in the `$SRSSITE/srsdb.i` file to the same directory. This is frequently used for file-per-entry databases such as PDB and FSSP. In this case, SRS Prisma will ensure that any datafiles, which would otherwise be overwritten, are archived as directed. It is recommended that the online index directory is not set to be the same as the offline index directory. If this is done SRS Prisma will not handle the archiving of the old index files correctly.

3.11.3 Installation Commands

Additional commands may be specified to run before and after installation of a library. One or more can be added using the appropriate list in the **Installation actions** panel:

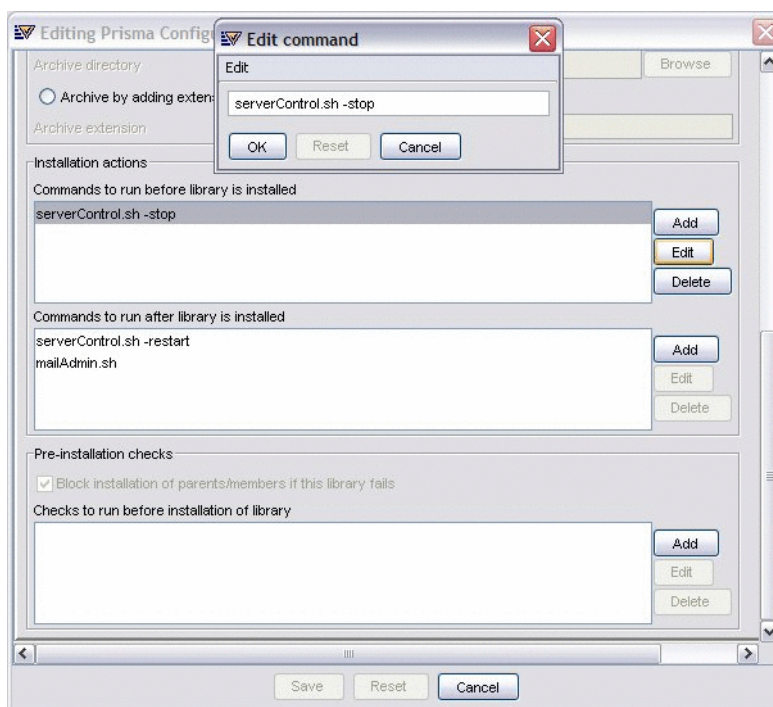


Figure 3.44 Specifying pre- and post-installation commands.

Note that the `%d` placeholder may be used in each case to be replaced by the offline data directory.

These commands may be specified manually using the `preInstallCommand` and `postInstallCommand` attributes of the Resource object:

```
preInstallCommand:{
    "serverControl.sh -stop"
}
postInstallCommand:{
    "serverControl.sh -restart"
    "mailAdmin.sh"
}
```

3.11.4 Installation Checks

Before installation of successfully updated libraries takes place, two different sets of checks take place.

3.11.4.1 Hierarchy Checking

The first check is whether the library is part of the parent/member hierarchy of a virtual or dependent library which failed. If it is, the library is not moved online, and any updated links involving it are rebuilt using online data. For instance, the member library UNIPROT-SWISSPROT will not be moved online if the virtual library UNIPROT failed. This is to ensure consistency between different libraries. If this behavior is not desirable, it can be suppressed in the virtual or dependent library by unchecking **Block installation of parents/members if this library fails**. (The corresponding Resource attribute is `blockParentInstallation`.)

3.11.4.2 Specified Checks

Secondly, a range of checks can be run on a library at install time. Unless they all pass, the library will not be installed automatically, and any links involving them will be rebuilt using online data to ensure consistency. Failed libraries can be installed using the `movePrisma` command (see Section 4.5, Moving Libraries Manually, page 142).



Note: It is important to realize that if a narrow installation window is required for the entire installation procedure (see Section 3.5.1, Pre-installation Checks, page 45), then if an individual library fails its specified check, then links may need to be rebuilt. This may be a time-consuming process, so the administrator must be aware that this may overrun the main installation window. It is strongly recommended that the conflicting requirements of installation windows and checks be carefully considered for each library before configuring both types of checks.

These checks can be added by clicking **Add** on the panel labelled **Checks to run before installation of this library** to open a dialog to edit the check:

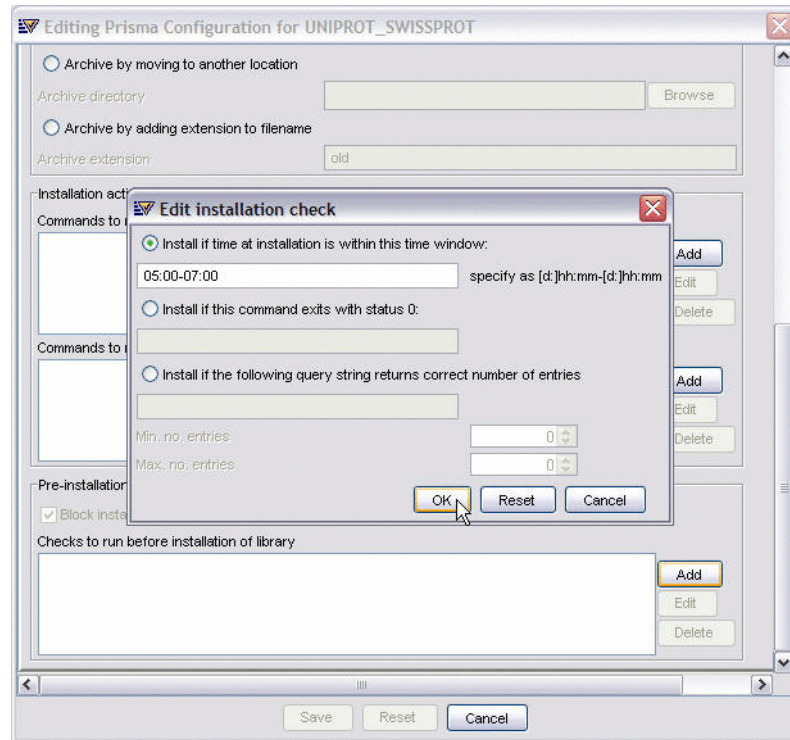


Figure 3.45 Adding an installation check.

Checks can be added manually as `PrismaInstallCheck` objects to the `installChecks` list:

```
installChecks:{
  $PrismaInstallCheck:{type:time
    installWindow:"05:00-07:00"
  }
}
```

3.11.4.3 Time-based Checking

A time-based check will mean that the library will only be installed if the install phase is executed within the specified time window. The time window is specified using the same format described in Section 3.5.1, Pre-installation Checks, page 45.

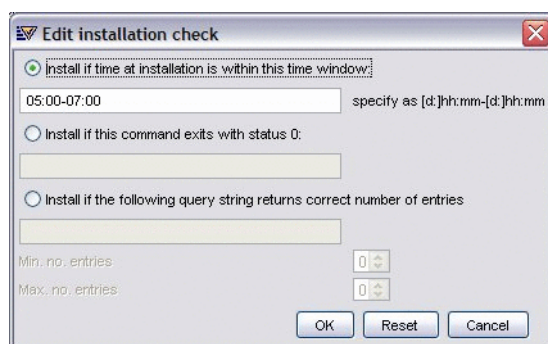


Figure 3.46 A time-based installation check.

The equivalent `PrismaInstallCheck` object is:

```
$PrismaInstallCheck:[  
  checkType:time  
  installWindow:"05:00-07:00"  
]
```

3.11.4.4 Command-based Checking

A command-based check will mean that the library will only be installed if the specified command exits with 0 (zero) status. The specified command is executed as a shell script.

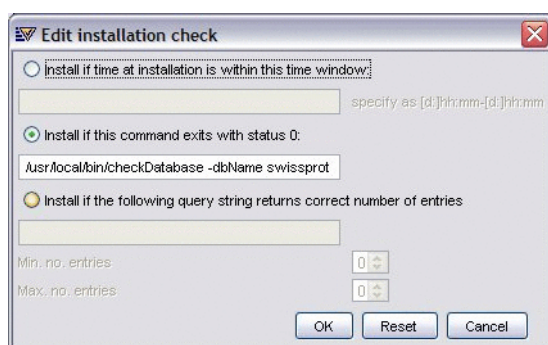


Figure 3.47 A command-based installation check.

The equivalent `PrismaInstallCheck` object is:

```
$PrismaInstallCheck: [checkType:command
  installCheckCommand:"/usr/local/bin/checkDatabase -dbName uniprot_swissprot"
]
```

3.11.4.5 Query-based Checking

A query-based check will mean that the library will only be installed if the specified query of the offline data set returns between the specified minimum and maximum number of entries:

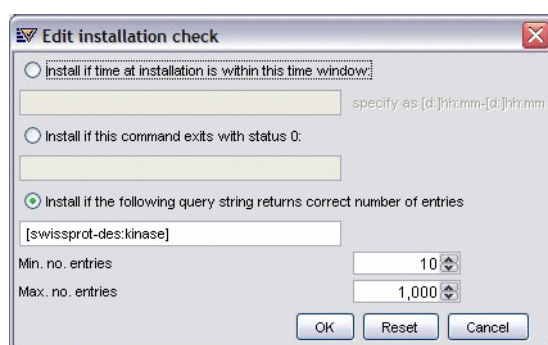


Figure 3.48 A query-based installation check.



Note: If the maximum number of entries is set to 0 (zero), no upper limit is assumed.

The equivalent `PrismaInstallCheck` object is:

```
$PrismaInstallCheck: [
  checkType:query
  installCheckQuery:"[uniprot_swissprot-des:kinase]"
  installCheckQueryMax:1000
  installCheckQueryMin:10
]
```

3.12 Schedule Settings

3.12.1 Scheduling for Automatic Updates

SRS Prisma is designed to be run automatically on a daily basis, and as such, allows flexibility in the extent of the check and update carried out on each day. For instance, it may be desirable to schedule the update of large data sets to take place over weekends.



Note: Schedules added in the following section only apply when `launchPrisma` is used to run Prisma (and not when `runPrisma` is used on the command line). Scheduling logic is only applied when one or more schedules have been specified. If no schedules are present for the library, normal checking takes place.

To add a schedule for a library, open the **Schedule settings** tab for a library, and click **Add** on the panel labelled **Schedules for automatic updates**:

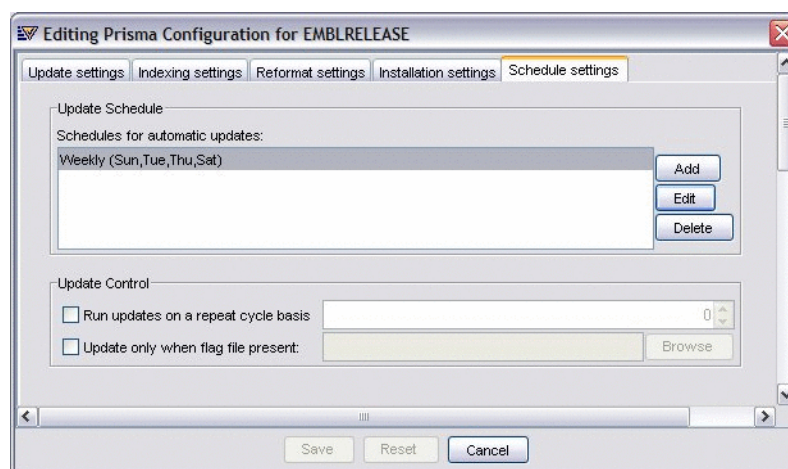


Figure 3.49 Adding a schedule.

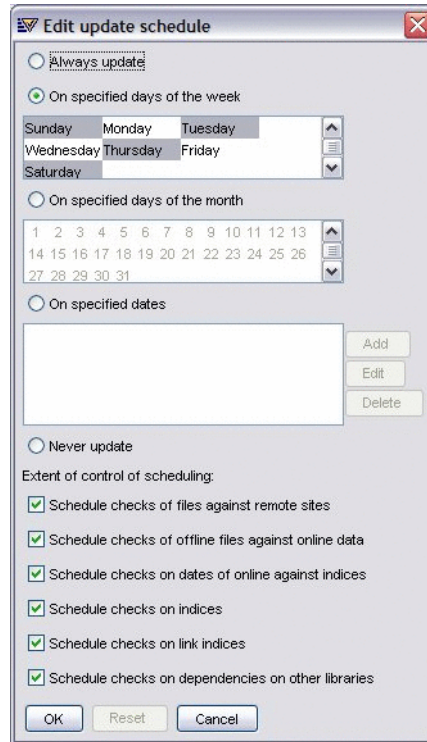


Figure 3.50 Editing a schedule.

Schedules can also be added manually as `PrismaSchedule` objects to the `Resource` `schedule` attribute:

```
schedule:{
  $PrismaSchedule:[scheduleType:weekly
    weekDays:{0 2 4 6}
    scheduleRemoteCheck:y
  ]
}
```



Important: When automatic configuration updates are run as part of the main Prisma process (see Section 8.2.2, Running as part of Prisma, page 224), library update schedules **MUST** include indexing on all days that Prisma runs. This is to support the forced reindexing of libraries that some configuration updates may required.

3.12.2 Schedule Frequency

Schedules can be set to update libraries on different time-scales. The appropriate option can be set using the **Edit Update Schedule** window, or by setting the `PrismaSchedule` object as appropriate.

always update

When set, the library will always be checked and updated if necessary. An example `PrismaSchedule` object would be:

```
$PrismaSchedule:[scheduleType:always
  scheduleRemoteCheck:y
]
```

on specified days of the week

When set, the library will only be checked if the current week day matches one of the selected days. An example `PrismaSchedule` object would be:

```
$PrismaSchedule:[scheduleType:weekly
  weekDays:{0 2 4 6}
  scheduleRemoteCheck:y
]
```

(note that `weekDays` is specified as a list of integers where 0 (zero) is Sunday)

on specified days of the month

When set, the library will only be checked if the current day of the month matches one of the selected month days. An example `PrismaSchedule` object would be:

```
$PrismaSchedule:[scheduleType:monthly
  monthDays:{1 15}
  scheduleRemoteCheck:y
]
```

on specified dates

When set, the library will only be checked if the current date matches one of the specified dates. The date can be specified in the form `YYYYMMDD`. An example `PrismaSchedule` object would be:

```
$PrismaSchedule:[scheduleType:dates
  scheduleRemoteCheck:y
  dates:{"20040503" "20040728"}
]
```

never update

When set, the library will always be checked and updated if necessary. An example `PrismaSchedule` object would be:

```
$PrismaSchedule: [scheduleType:never  
  scheduleRemoteCheck:y  
]
```

3.12.3 Schedule Control

The extent of control of the schedule on the SRS Prisma check process can be controlled by setting the appropriate stage to be controlled.

Schedule checks of files against remote sites

If this option is set, the library will only be checked remotely if the schedule is satisfied. The equivalent attribute of the

`PrismaSchedule` object is `scheduleRemoteCheck` (set to `y` or `n`).

Schedule checks of offline files against online data

If this option is set, the offline data files will only be checked if the schedule is satisfied. The equivalent attribute of the `PrismaSchedule` object is `scheduleOfflineCheck` (set to `y` or `n`).

Schedule checks on dates of online against indices

If this option is set, the online data files will only be checked if the schedule is satisfied. The equivalent attribute of the `PrismaSchedule` object is `scheduleOnlineCheck` (set to `y` or `n`).

Schedule checks on indices

If this option is set, the indices will only be checked if the schedule is satisfied. The equivalent attribute of the `PrismaSchedule` object is `scheduleIndexCheck` (set to `y` or `n`).

Schedule checks on link indices

If this option is set, link indices involving the library will only be checked if the schedule is satisfied. The equivalent attribute of the `PrismaSchedule` object is `scheduleLinkCheck` (set to `y` or `n`).

Schedule checks on dependencies on other libraries

If this option is set, the library will only be checked against 'parent' libraries if the schedule is satisfied. The equivalent attribute of the `PrismaSchedule` object is `scheduleDependencyCheck` (set to `y` or `n`).

3.12.4 Advanced Scheduling Control

In addition to the normal Prisma scheduling mechanism, there are some advanced control mechanisms that may be useful in certain circumstances.

The first control allows a library to be checked every time Prisma is run, but only updated every *n* times. The update frequency can be set in the **Installation settings** tab, or using the `Resource updateRepeatCycle` attribute.

Secondly, a library can be set to be checked every time Prisma is run, but only updated when a specified file is present locally. The file can be specified using the **Installation settings** tab or using the `Resource updateFlagFile` attribute.

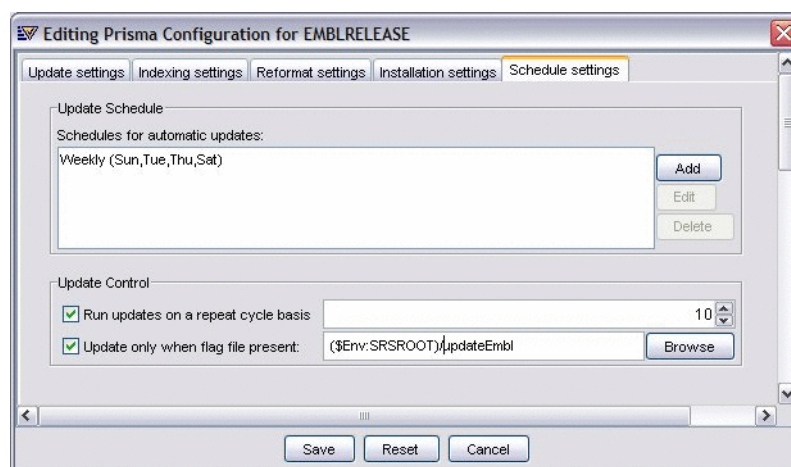


Figure 3.51 Setting advanced scheduling control.



Note: When considering data sets for which a separate release set and update set exist (e.g. GENBANKRELEASE and GENBANKNEW), care must be taken when scheduling to ensure that updates for a new release are not downloaded until the new release has been downloaded as well. Failure to do so may lead to an incomplete data set. It may be advisable to turn off remote updating for both libraries close to an update until local systems are ready for the update.

3.13 Library-specific batch queue commands

In addition to being able to specify batch queuing commands for individual stages of the update process, SRS Prisma 4.1 also allows batch queue commands to be specified for individual libraries. This allows specific queues to be used for specific tasks that may have specific resource requirements (e.g. execution on a host which has the requisite software available). When batch queuing is enabled (see Section 3.3.2, Using a Batch Queue System, page 35), the **Queue Settings** tab is present in the Library Configuration Dialog:

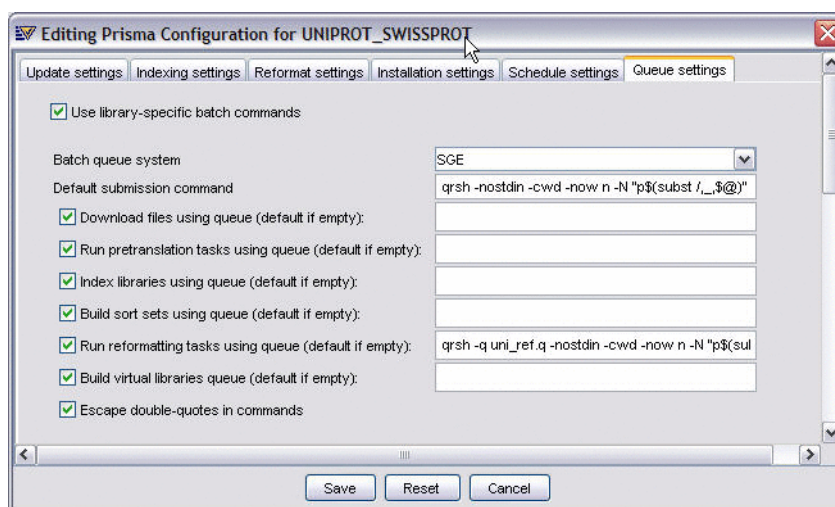


Figure 3.52 Enabling library-specific batch queue commands.

To enable library-specific batch queue commands, check “Use library-specific batch commands”. The default command is inherited from the system setting, but can be

changed as required. In addition, specific stages can be changed so they do not use batch queuing (by unchecking the stage in question), or use a different command (by entering a new command for the stage in question). In Figure 3.52, an alternative commands has been supplied for the reformat stage.

To configure this behavior directly in the `Resource` object, the following attributes should be set:

`Resource.useSpecifiedBatch`

should be set to `y` or `n`

`Resource.batchQueue`

should contain an `PrismaBatchQueueSystem` object (see Section 3.3.2.4, Configuration of Batch Queue System, page 38).

3.14 Dependent Libraries

Some data sets are not downloaded or copied from remote locations, but are derived indirectly (for example, by running SRS queries on existing libraries). Once these data sets have been created, they themselves can be indexed in SRS, or otherwise processed. This section describes the creation, use and limitations of dependent libraries.

In addition to updating normal SRS libraries, SRS Prisma also includes the concept of a dependent library. In a typical SRS Prisma run, 'normal' libraries are updated in the first 'library level' of updating. Dependent libraries are updated in successive levels if their parent (normal or dependent) has itself been successfully updated. They may also be re-indexed if their indices are out of date, even if the parent has not been updated.

3.14.1 Configuring a Library as Dependent

To specify a library as dependent, open the **Update settings** tab for the library in question, and select **Update when other libraries are updated**. The 'parent' libraries upon which the library depends can then be selected:

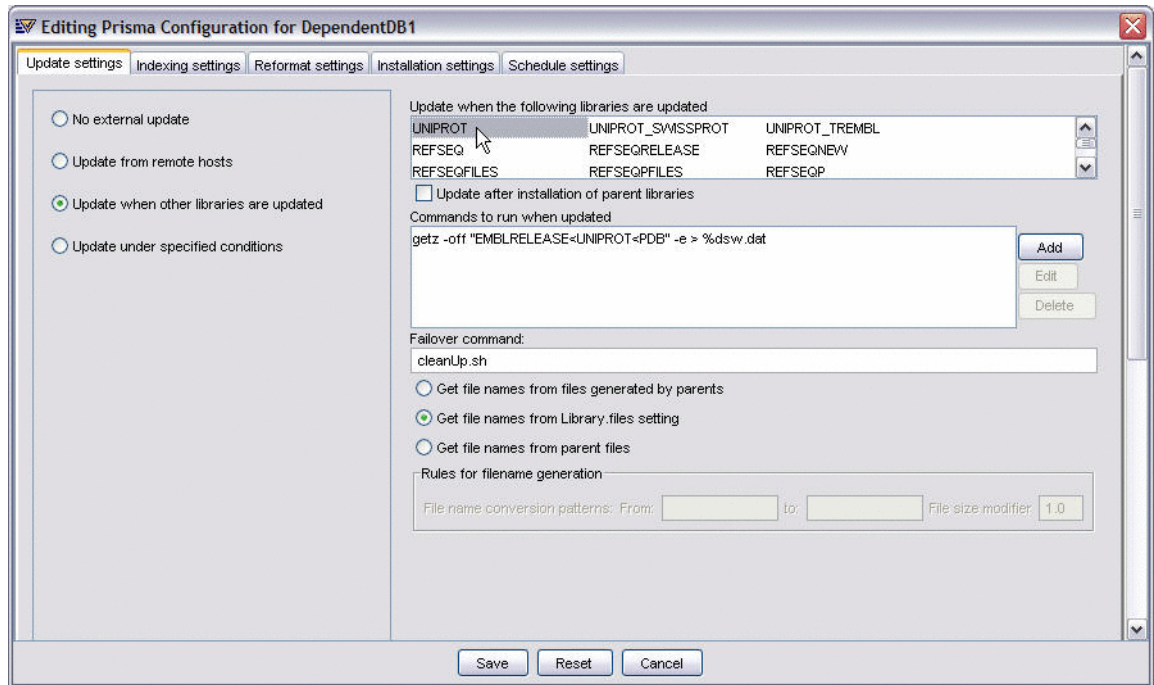


Figure 3.53 Setting a library as dependent.

To specify a library as dependent directly using the `Resource` object, set the `updMethod` attribute to `create` and specify the list of parent libraries as a list using the `dbDependOn` attribute. For example:

```
$DEP_DB = $Resource:[DepDb
  updMethod:create
  dbDependOn: { "EMBLRELEASE" "UNIPROT_SWISSPROT" "PDB" }
]
```

In a typical SRS Prisma run, 'normal' libraries are updated in the first 'library level' of updating. Dependent libraries are updated in successive levels if their parent (normal or dependent) has itself been successfully updated. They may also be re-indexed if their indices are out of date, even if the parent has not been updated.

3.14.2 Configuring a Library as Dependent on Online Data

Dependent libraries are normally updated after their parents are updated, but before they are moved online. Alternatively, they can be updated after the parents are successfully moved online, and then may become the trigger for another round of updating if any other libraries depend on them. They are moved online in a subsequent move phase if successfully updated. To set a library to depend on the move phase of its parents, check **Update after installation of parent libraries** or specify the `updMethod` attribute of the `Resource` object as `createfromonline`:

```
$DEP_DB = $Resource:[DepDb  
  updMethod:createfromonline  
  dbDependOn: { "EMBLRELEASE" "UNIPROT_SWISSPROT" "PDB" }  
]
```

3.14.3 Specifying Preprocessing Commands

If a library is defined as dependent, no attributes of the `$Resource` class object relating to downloading files, filename patterns, and so on, should be set (any that are set will be ignored). However, typically the preprocessing commands associated with the library should contain the commands required to create all the datafiles for the library, provided if they have not been created by any of the parents (at the reformat stage and so on). This command cannot use filename substitution, and will be called just once, after some or all of the processing of the parent(s) has been completed. To specify the commands used, use the **Commands to run when updated** panel of the **Update settings** tab:

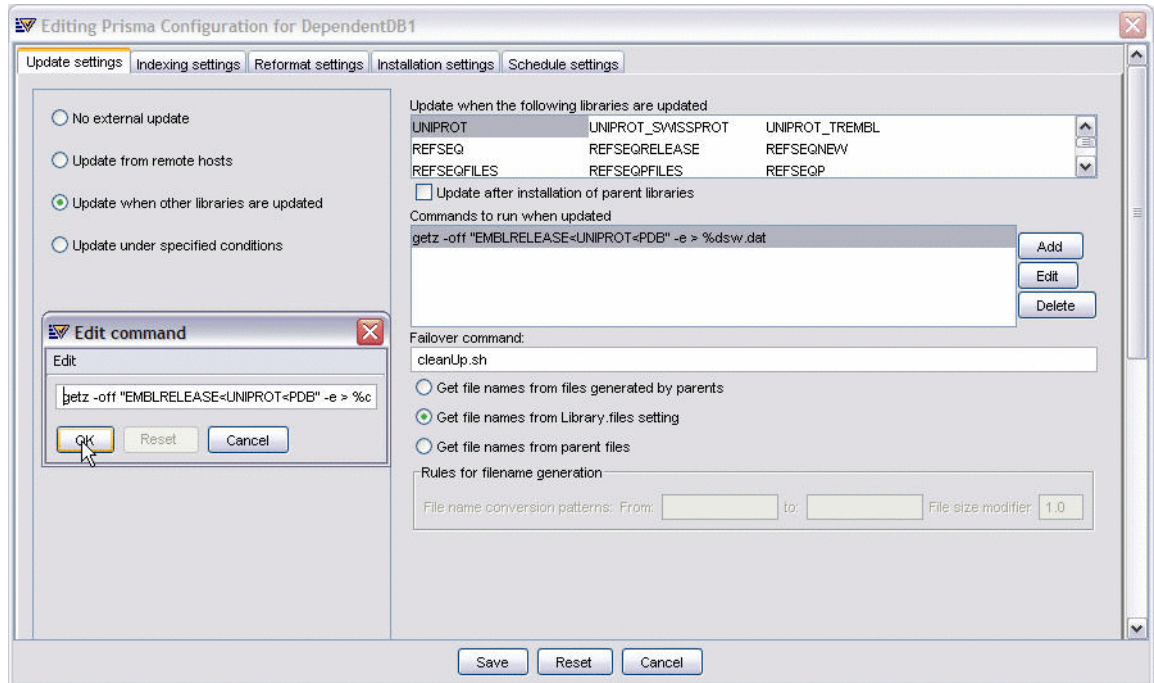


Figure 3.54 Adding a preprocessing command.

or the `unpackCommand` attribute of the `Resource` object (see Section 3.8.1.13, Specifying Preprocessing Commands, page 72).

It should be noted that preprocessing commands (or, indeed, postprocessing commands of the parents) will be called while some or all of the libraries on which it depends are still offline. The commands it contains must identify which libraries are offline by checking the relevant offline data directories for datafiles, and/or running any SRS queries in an offline mode, for example, `getz -off`. Using the `-off` argument should be safe, since if the library does not have any offline indices, it will automatically switch to the online copies. The same applies for the `off` argument of the `Icarus $Session.query` method.

In addition to the standard placeholders, preprocessing commands accept the `%DBNAME%` placeholder. This specifies the online or offline data directory for the specified library, for example, `%UNIPROT_SWISSPROT%`. SRS Prisma correctly selects between online and offline directories based on whether the parent library was updated. Only library names from the `dbDependOn` list will be recognized.

In addition, a failover command may be specified that can be run if one or more pre-processing command fails. This allows processes started by the pre-processing commands to be stopped cleanly and any clean-up commands to be executed. A failover command may be entered in the **Failover command** box on the **Update settings** panel or set directly using the `Resource unpackFailureCommand` attribute.

3.14.4 Non-SRS Dependent Libraries

A dependent library can be set as non-SRS so that it can be used as a mechanism to run commands in response to the updates of other libraries. See Section 3.9.2, Non-SRS Libraries, page 87 to find out more about non-SRS libraries.

3.14.5 Uses of Dependent Libraries

Dependent libraries can be extremely flexible, if used judiciously, allowing extremely complex data management strategies to be implemented with SRS Prisma. In particular, specification of `noSRS:yes` for a dependent library allows the execution of additional or more complex post-processing commands.

Typically, dependent databases of this type are used for a number of different purposes. These include:

- Indexing of files downloaded and unpacked from a tarball by a `noSRS` parent (for example, `PATHWAYTAR` and `PATHWAY`).
- Indexing the datafiles of a parent in a different way (for example, with a different parser). See the `uniest.it` and `uniseq.it` files in the distribution.
- Production of FASTA format files and BLAST indices from the datafiles or from queries involving multiple parents.

3.14.6 Limitations of Dependent Libraries

3.14.7 Building Dependent Libraries Manually

A dependent library will only be rebuilt if one of its parent library is being downloaded or indexed on a given invocation of SRS Prisma. If a parent library is rebuilt manually, or if the dependent library is excluded by one of the command line options of SRS Prisma. For example:

```
% runPrisma -l parentlib
```

then SRS Prisma will not detect, on later invocations, that the parent library has been rebuilt. When this occurs, the data for the dependent library should be recreated manually. This is done by calling the command specified as `unpackCommand`. The dependent library can then be re-indexed using SRS Prisma or `srscheck`. It is not practical to regenerate dependent libraries automatically with SRS Prisma, as the `unpackCommands` for dependent libraries frequently require specific data, from their parents, to be in specific locations, and this is outside of SRS Prisma's sphere of control.

3.14.8 Dependent Library File Lists

If a dependent library uses datafiles generated by a parent, then in a normal SRS Prisma run, when it determines that a dependent library should be created, SRS Prisma will also determine which files are available in the offline directory and write commands for indexing and moving them as appropriate.

However, if the dependent library creates new files itself, SRS Prisma does not have access to the list of files created by the preprocessing commands and hence it is not able to move the datafiles online. Installing the datafiles should be handled by the installation commands (see Section 3.11.1.3, Use Specified Commands for Installation (command), page 104 – note that filename substitution is not possible). Alternatively, the online and offline data directories can be set to be the same. SRS Prisma should be able to install any SRS index files that it generates for this databank in the normal way. This also has the knock-on effect of limiting the parallelization options available for a dependent database, since the names and

sizes of datafiles are not known when SRS Prisma writes the commands needed for indexing.

However, SRS Prisma can be provided with more information on files used by a library, as described in the next section.



Note: Although the following options can be extremely useful, it is recommended that they are used with care, particularly where generated files are large and subject to change, and that where possible, files used by dependent databases are pre-generated by parent databases.

3.14.8.1 Using the Library.files Attribute

This mechanism uses the files attribute of the `$Library` class object. This is an Icarus list of `LibFile` objects, specifying a basename. It is normally used in conjunction with the `$LibFormat.fileType` information to determine which files should be indexed by SRS. However, SRS Prisma can use this filename information to construct a list of filenames for use in parallelization. In addition, the `$LibFile.size` attribute can also be set to provide an approximate size in kb which will allow SRS Prisma to provide `filesSize` or `chunkSize` parallelization. To activate this mechanism, select the **Get file names from Library.files setting** option on the **Update settings** tab:

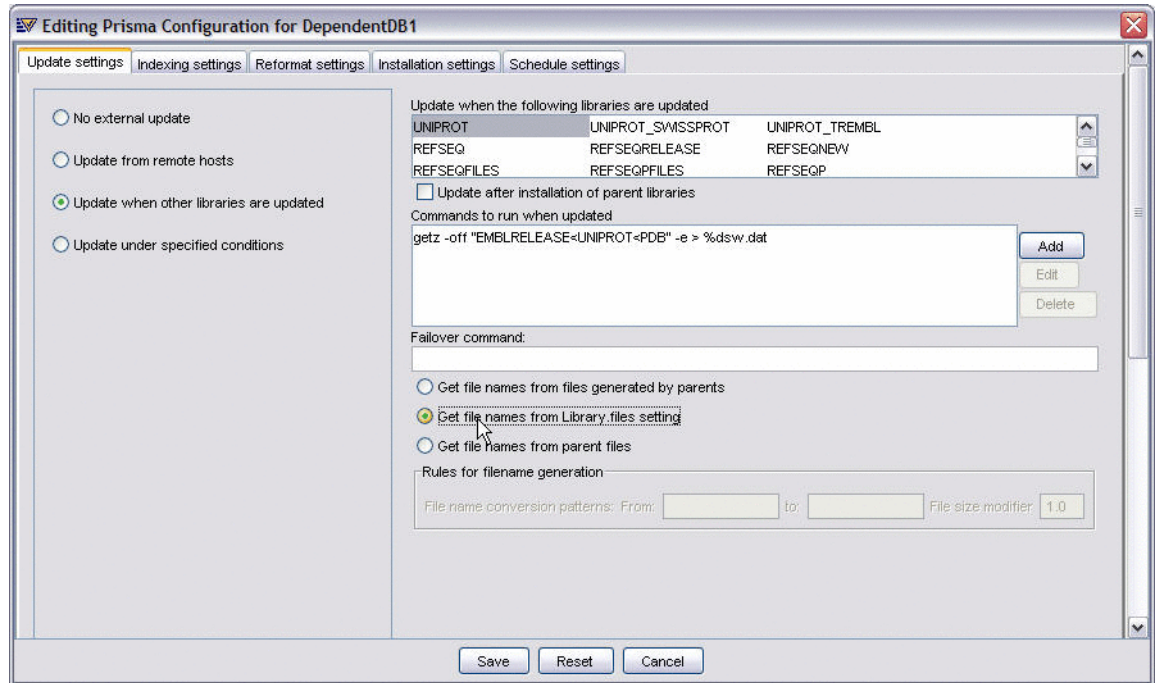


Figure 3.55 Using Library.files to get file information.

To set this directly, set the Resource usesFiles attribute to 'yes':

```
$DEP_DB = $Resource:[DepDb
  updMethod:createfromonline
  dbDependOn: {"EMBLRELEASE" "UNIPROT_SWISSPROT" "PDB"}
  usesFiles:yes
]
```



Note: If the named files are not available, indexing will not proceed correctly.

3.14.8.2 Inheriting File Information from Parents

This mechanism allows SRS Prisma to determine a list of files, based on the files available to the parent library. For example, if SRS Prisma is used to derive a file called `em_fun` from the `EMBLRELEASE` file, `fun.dat`, then SRS Prisma can use the

filename and size information, available to the parent, to create a file list for the dependent database. When this mechanism is used, SRS Prisma needs to be provided with regular expressions to convert from the parental filename to the derived filename, and size multipliers to indicate the relative size of the derived file. This can be set by selecting the option **Get file names from parent files** and entering the regular expressions mapping between (this are analogous to those used from remote to local file conversion), and a value for the file size conversion factor:

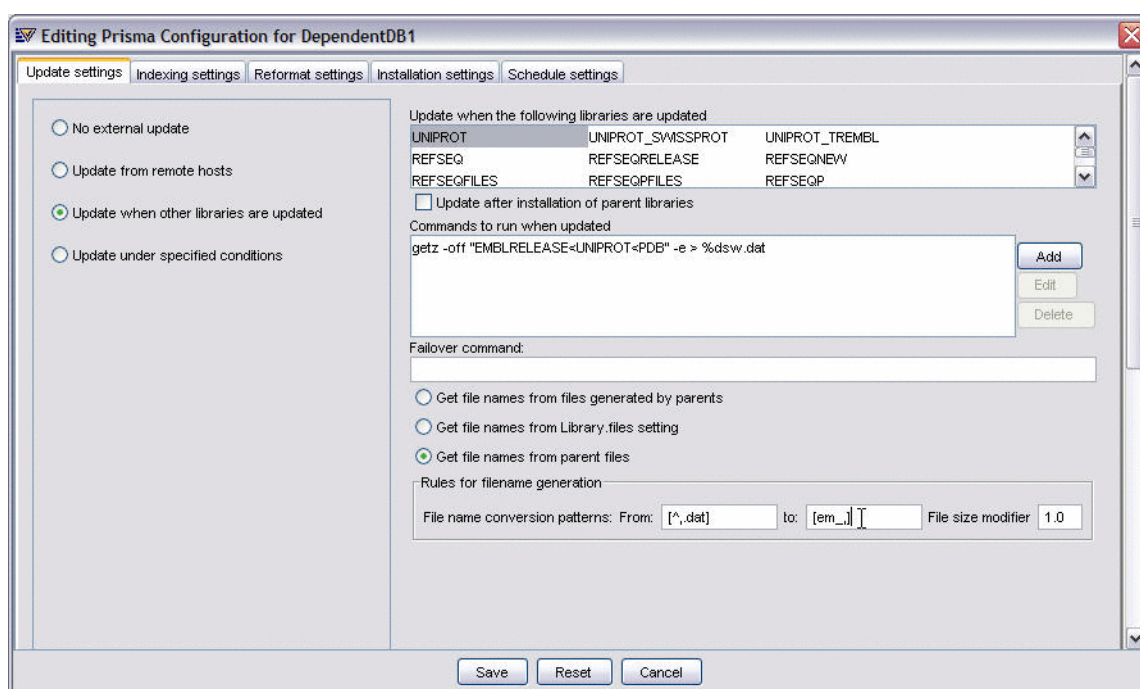


Figure 3.56 Inheriting file information from parents

To set this directly using the `$Resource.inheritsFiles` attribute should be set to yes. The attributes, `$Resource.inheritFileSize`, `$Resource.inheritFilePatternFrom` and `$inheritFilePatternTo` can be used to provide information on how the files in the parent compare to those in the child. e.g.

```
$EMBLDEP_Res=$Resource: [
..
updMethod:create
dbDependOn: "EMBLRELEASE"
```

```
unpackCommand:"find /path/to/embl -name "*.dat" -exec createFasta.sh {} \;"
inheritFiles:yes
inheritFilePatternFrom:"[^,].dat]"
inheritFilePatternTo:"[em_,]"
inheritFileSize:0.5
]
```

This will take a file named `fun.dat`, of size 100 Kb, from the parent and supply a file description of name `em_fun` and size 50 Kb to the child database.

In this case, `inheritFilePatternFrom` and `inheritFilePatternTo` are strings of the form `[pattern1,pattern2]` that can contain multiple regular expressions separated by strings.



Note: The filename, at least, must be accurate (it must be produced before indexing), otherwise the indexing of this database may fail. In addition, the file list provided by both of these mechanisms is also used to install the flat-files online.

3.15 External Update Triggers

In addition to a library update being triggered by new or updated files (local or remote) or by the update of a parent, a library can also be triggered to be updated by an external factor. A list of trigger conditions can be specified, of which at least one must be satisfied for the update to take place. To define a library as being updated on the basis of an external trigger, select **Update under specified conditions** from the **Update settings** tab, and click **Add** on the panel labelled **Update under the following conditions** to add one or more triggers:

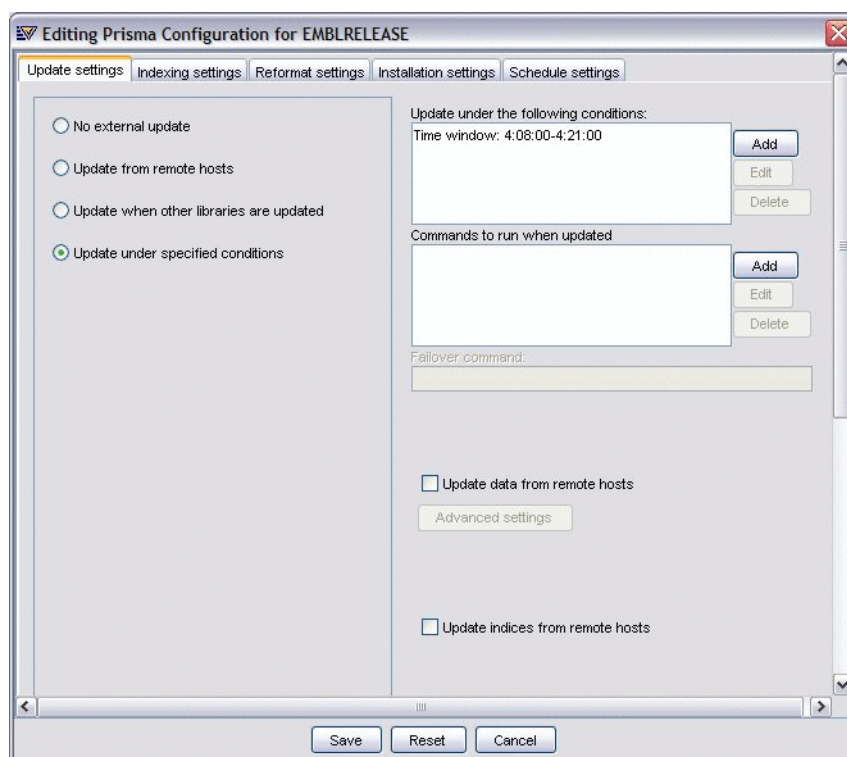


Figure 3.57 Adding an external trigger.

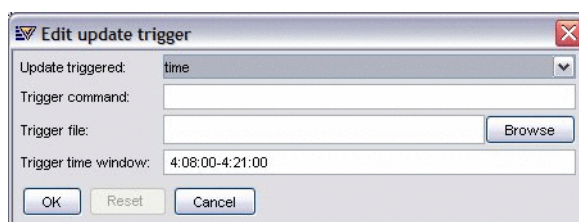


Figure 3.58 Editing an external trigger.

The equivalent Resource object has `updMethod` set to `trigger`, and one or more `PrismaUpdateTrigger` objects in the `triggers` attribute:

```
$TRIG_Res = $Resource:[TriggerDb
    updMethod:trigger
```

```

    updateTrigger:{
      $PrismaUpdateTrigger:[checkType:time
        updateWindow:"4:08:00-4:21:00"
      ]
    }
  ]
}
]

```

The trigger can be defined using the trigger dialog, and can be one of four types:

always

When run, the library is always updated.

command

The library is updated if the specified shell command returns with exit status 0.

file

The library is updated if the specified file is present on the server.

time

The library is updated if the current time falls within the specified time window. The time window uses the format used in Section 3.5.2, Installation Options, page 47.

The `PrismaUpdateTrigger` object can be configured directly, and has the following attributes:

checkType

type as above (always, command, file or time)

updateWindow

string representing time window

updateCheckCommand

shell command to run

updateCheckFile

path to file to check locally

The processes run during the update depend on how the library is configured, but can be thought of as a forced update. If remote data or index files are specified, they will be downloaded. Otherwise, any associated pre- or post-processing commands will be run. If the library is an SRS library, then indexing will also be carried out.

Remote data and index files can be specified as for 'normal' libraries, using the **Update settings** tab:

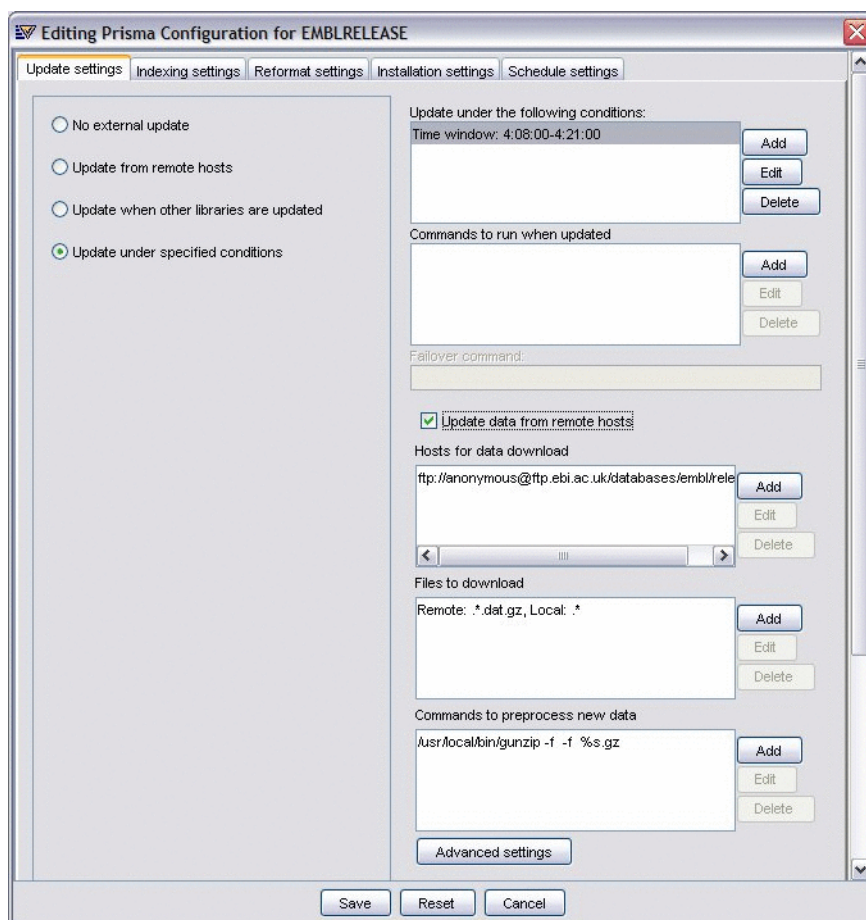


Figure 3.59 Specifying remote data and index files.

(see Section 3.8.1.1, Remote Data Updates, page 56 for more details). All files from the remote site will be downloaded every time that the conditions are satisfied on running SRS Prisma.

CHAPTER

4

RUNNING SRS PRISMA

4.1 Introduction

Once SRS Prisma is installed, it can be quickly and easily run to check and update any installed SRS databases.

There are two ways to launch the SRS Prisma update process:

- Manually
- Automatically



Note: Typically, SRS Prisma is not run manually because it has been designed to be run as an automatic process on a regular basis.

4.2 Running SRS Prisma Manually

SRS Prisma can be run manually from the command line.

The check and update process in SRS Prisma is controlled by a single application, `$SRSETC/runPrisma`. This runs the staged checking and update process (see Section 1.2.2, The Prisma Staging Process, page 3); and, also compiles update and quality reports. It is referred to as a ‘manual’ run. It can be run simply from the command line as follows:

```
% runPrisma
```

While the SRS Prisma process is running, a number of messages are displayed about the progress of the update process. When the process has finished successfully, a full report is available from the ‘current’ subdirectory of the HTML report directory specified during the install process. In addition, interim ‘snapshot’ reports are written at the end of each successful update stage. These reports can be accessed with the **View latest report** button on the main report calendar page, but note that these reports are not archived. More details on obtaining and interpreting SRS Prisma reports are available in Chapter 5 SRS Prisma Update Reports.



Note: When SRS Prisma is used to update an installation, it is not recommended that `srsccheck/srsdo` are used. This is because `srsccheck` will not honour specialized

Prisma configurations such as 'switchlinking' and archiving. To carry out specific local updating, it is recommended that `runPrisma -local` is used instead (see below).

4.2.1 Option Flags

A collection of command line option flags is available for use with `runPrisma`, which alter the behavior of SRS Prisma.

Table 4.1 Command line option flags controlling checking

Flag	Type	Default Value	Description
-v	Boolean	FALSE	Display verbose output (see Section 3.4.1, Show Verbose Output, page 43)
-d	Boolean	FALSE	Display diagnostic output (see Section 3.4.2, Show Debugging Output, page 43)
-l	string	""	Restricts libraries checked to those named (specified as a space-separated list). Note: SRS Prisma will normally check all libraries on an installation. (see Section 3.4.10, Libraries to Check, page 45)
-L	string	""	Excludes the named libraries from the range of checking. (see Section 3.4.11, Libraries to Exclude from Checking, page 45)
-g	string	""	Restricts checking to libraries in the named groups. (see Section 3.4.12, Groups to Check, page 45)

-G	string	""	Excludes libraries from the named group from the range of checking. (see Section 3.4.13, Groups to Exclude from Checking, page 45)
-f	string	""	Index specified fields only.
-force	Boolean	FALSE	Forces all libraries in the range of checking to be updated, regardless of status (this includes remote updates where possible).
-forceLinks	Boolean	FALSE	Forces all links in the range of checking to be updated, regardless of status. Note that libraries will also be updated if necessary.
-forceReformat	Boolean	FALSE	Do not carry out normal checks, but force reformat commands to be executed for the specified libraries (see Section 3.4.6, Do Not Check links, page 43).
-nolinks	Boolean	FALSE	Suppress the building of link indices. Note that if these are excluded, links may become out-of-date and return the wrong results. (see Section 3.4.6, Do Not Check links, page 43)
-links	Boolean	FALSE	Checks links between specified libraries and all others on the system. Note that normally, SRS Prisma will build link indices between all libraries in the range of checking. (see Section 3.4.5, Check All Links, page 43)
-local	Boolean	FALSE	Suppress remote checking and downloading. (see Section 3.4.3, Carrying Out Local Check Only, page 43)

-downloadOnly	Boolean	FALSE	Restricts checking and updating to remote updates only (no indexing or installation of files will be carried)
-checkonly	Boolean	FALSE	Check for required updates, but do not carry out those updates.
-time	string	cre	Use specified timestamp for local file checking (cre, mod or acc). (see Section 3.4.9, File Timestamp to Use, page 44)

Table 4.2 Command line option flags controlling execution of the run

Flag	Type	Default Value	Description
-run	string	Manual	Name of run
-restart	Boolean	FALSE	Restart previous failed Prisma run
-interactive	Boolean	FALSE	Run in interactive mode, and pause for keyboard input at different stages of the process. This allows troubleshooting by checking files etc.
-autoRepeats	int	1	Number of times to repeat process in case of failure (see Section 3.3.1.2, Max. Repetitions of Failed Updates, page 34)
-numProcs	int	1	Number of processes to run in parallel (see Section 3.3.1.4, Maximum Parallel Processes, page 34)
-parMake	Str	\$\$SRSEXE/gmake	Make command to use (see Section 3.3.1.3, Make Command, page 34)

Table 4.3 Command line option flags controlling reporting

Flag	Type	Default Value	Description
-reportDir	string	\$SRSWWW/prisma	Write reports to specified directory (see Section 3.6.2, Location for HTML Reports, page 49)
-noQuality	Boolean	FALSE	Do not create quality report (see Section 3.6.5, Generate Quality Reports, page 49)
-noTrace	Boolean	FALSE	Do not create trace reports (see Section 3.6.6, Generate Trace Reports, page 50)
-archive	Boolean	FALSE	Archive reports (default for non-manual runs)

Table 4.4 Command line option flags controlling advanced behavior

Flag	Type	Default Value	Description
-mainTarget	string	firstTarget	Target to build for each stage
-splitRun	string		Execute specified stage of split run ('download' or 'index'). (see Section 4.6.1, Split Runs, page 144)
-saveState	Boolean	FALSE	Read state from previous run (see Section 4.6.2, Run State Preservation, page 145)
-ignoreFailures	Boolean	FALSE	This option is for use with -restart, and does not restart update stages that completed with failures.

-nomove	Boolean	FALSE	Do not automatically install new data/indices (see Section 4.5, Moving Libraries Manually, page 142)
-noWarnings	Boolean	FALSE	Suppress warnings from srsbuild (see Section 3.3.1.6, Suppress Warnings from srsbuild, page 35)
-saveUnc	Boolean	FALSE	Do not remove uncompressed files (see Section 3.5.5, Do Not Delete Uncompressed files, page 47)
-keepLinks	Boolean	FALSE	Do not remove offline symbolic links (see Section 3.4.8, Preserve Offline Symbolic Links, page 44)
-completed	Boolean	FALSE	Honour completed flags (see Section 3.4.7, Use COMPLETED Flag Checking, page 44)
-updateConfigFiles	Boolean	FALSE	Update configuration files
-noUpdateConfigFiles	Boolean	FALSE	Do not update configuration files
-configName	String	""	Configuration files set to update

4.3 Running SRS Prisma Automatically

Typically, SRS Prisma is run as an automatic process on a regular, sometimes daily, basis. To facilitate this, the script `$SRSETC/launchPrisma` has been provided. It can be launched from the command line, or from a batch or cron process:

The script `launchPrisma` sets the SRS environment to that of the current installation, which is set automatically during the install process, but can be altered by manually editing the script, and then launches the `runPrisma` process as an 'automatic' run. This runs SRS Prisma as described in Section 4.2, Running SRS Prisma Manually, page 134, but the reports produced are automatically archived for future reference (see Chapter 5 SRS Prisma Update Reports for more details).

4.4 During a SRS Prisma Run

When an SRS Prisma run is in progress, a large number of files are needed to monitor the progress of the update and provide information during reporting. These are stored in the directory structure beneath `$SRSPRISMA/prisma`, in directories pointed to by the environment variables `$SRSPRISMA/automatic` and `$SRSPRISMA/manual`.

To allow the flags for a manual run to exist alongside an automatic run, which can be useful for running quick manual updates after an automatic update has failed, these environment variables can be set to different directories. For an automatic run, `$SRSPRISMA/automatic` is set to `$SRSPRISMA/automatic` and `$SRSPRISMA/automatic/flags` is set to `$SRSPRISMA/automatic/flags`. For a manual run, `$SRSPRISMA/manual` is set to `$SRSPRISMA/manual`. `$SRSPRISMA/manual/flags` is symbolically linked to `$SRSPRISMA/manual/flags`, which is also used by the `srsccheck` application.



Note: It is also possible to define other runs so that multiple update processes, by either setting `$SRSPRISMA/automatic` and `$SRSPRISMA/manual` manually, or by launching `runPrisma` with the `-run` argument. This may be useful for running multiple non-exclusive update jobs. However, it is generally recommended that `runPrisma` only be used for launching 'manual' jobs and `launchPrisma` for 'automatic' jobs.

Once an SRS Prisma process has been started, a lock file (`PRISMA_RUNNING`) is created in `$SRSPRISMA/`. The presence of this file prevents any other SRS Prisma processes from starting, and also prevents the SRS update processes `srsccheck` and `srscdo` from running. The reason for this is that parallel SRS indexing processes of this kind can frequently work against each other and cause problems with the correct execution of indexing.

However, this default behaviour can be changed to allow SRS Prisma to check whether the process that created the lock file is still running on the current host. If not, the lockfile is removed and the process can continue. Note that this behaviour may not be desirable if Prisma can be launched from multiple hosts. The configuration of this behaviour is described in Section 3.8.1.3, Specifying Remote Locations, page 56.



Important: When SRS Prisma is used to update an installation, it is not recommended that `srsccheck`/`srscdo` are used. This is because `srsccheck` will not honour spe-

cialized Prisma configurations such as 'switchlinking' and archiving. To carry out specific local updating, it is recommended that `runPrisma -local` is used instead.

4.4.1 Stopping SRS Prisma

Once SRS Prisma has started running, it is usually advisable to allow it to complete. However, a utility has been supplied to stop a Prisma run quickly and cleanly. To stop a Prisma run, run:

```
% stopPrisma -run <runName>
```

where `runName` is the name of the run to stop (usually automatic or manual).

This script finds the main `runPrisma` process, and terminates all processes associated with it. Note that if processes are executed on multiple hosts using a batch queue system, this script may not successfully halt all queued processes and care should be taken to clean up any additional processes.

4.4.2 Restarting SRS Prisma

SRS Prisma has been designed to behave in a robust manner, and to update an installation as far as possible with the resources available, responding to process failures where possible. Typically, if any part of an SRS Prisma update process fails, the next run of SRS Prisma will attempt to continue the update process as best it can; including processing of offline data, and installation of complete offline index and data sets.

However, there are occasions when the SRS Prisma process fails to complete the majority of its update process. Typically, this is due to a hardware problem such as network outage, server reboot, or lack of file space. In a situation such as this, it is possible that a substantial amount of indexing has been carried out, but not enough to produce complete indices that can be installed. In this case, it is desirable to be able to stop the SRS Prisma process (if it has not already started) and restart it at the points where it failed, or failed to complete.



Important: When using restarting SRS Prisma, it is critical that all processes from previous failed runs are killed. Generally, killing the top level process such as `launchPrisma` or `runPrisma` may not always kill all child processes. It is recommended that `stopPrisma` is used, or processes are manually cleaned up. Failure to do so will lead to unpredictable updating and reporting results.

SRS Prisma uses a checkpointing procedure to record each stage of the update process, so that the restart process can restart at the appropriate point according to where failure occurred. To restart a Prisma run:

```
% restartPrisma -run <runname>
```

where `<runname>` is either `manual`, or `automatic`, depending on whether the job was launched from `runPrisma` (`manual`) or `launchPrisma` (`automatic`).



Important: When restarting SRS Prisma, previously supplied command line arguments are **not** recalled, and must be supplied on the command line as before (`restartPrisma` passes all command line arguments to `runPrisma`).

By default, `restartPrisma` will find the first at which failure occurred and restart that process. However, for runs that were interrupted but had failures before the interruption, it is possible to ignore failures and simply restart at the last point before the interruption. To do this, specify the `-ignoreFailures` flag:

```
% restartPrisma -run <runname> -ignoreFailures
```

4.5 Moving Libraries Manually

SRS Prisma will automatically move successful libraries online, except in one of the following situations:

- the criteria for carrying out the move phase have not been satisfied.
- the criteria for moving the library online have not been satisfied.
- a dependent library or virtual library that depends on the library has failed.
- the `-nomove` flag has been specified.

To move libraries online manually, the `movePrisma` script has been provided. This will find libraries that have been blocked from being moved online, rebuild links involving these libraries (if necessary) and move them online. `movePrisma` accepts the command-line flags shown in Table 4.5 on page 143:

Table 4.5 Command line option flags controlling advanced behavior

Flag	Type	Default Value	Description
-l	string	""	Restricts libraries checked to those named (specified as a space-separated list). Note: SRS Prisma will normally check all libraries on an installation. (see Section 3.4.10, Libraries to Check, page 45)
-L	string	""	Excludes the named libraries from the range of checking. (see Section 3.4.11, Libraries to Exclude from Checking, page 45)
-g	string	""	Restricts checking to libraries in the named groups. (see Section 3.4.12, Groups to Check, page 45)
-G	string	""	Excludes libraries from the named group from the range of checking. (see Section 3.4.13, Groups to Exclude from Checking, page 45)
-forceMove	Boolean	FALSE	Force completed libraries to move online regardless of pre-installation checks (see Section 4.5, Moving Libraries Manually, page 142)
-numProcs	int	1	Number of processes to run in parallel (see Section 3.3.1.4, Maximum Parallel Processes, page 34)

<code>-parMake</code>	Str	<code>\$SRSEXEC/gmake</code>	Make command to use (see Section 3.3.1.3, Make Command, page 34)
<code>-v</code>	Boolean	<code>FALSE</code>	Display verbose output (see Section 3.4.1, Show Verbose Output, page 43)
<code>-d</code>	Boolean	<code>FALSE</code>	Display diagnostic output (see Section 3.4.2, Show Debugging Output, page 43)

Note that while the main installation checks will not be carried out (see Section 3.5.1, Pre-installation Checks, page 45), per-library checking (see Section 3.11.4, Installation Checks, page 109) will still be carried out. This can be overridden using the `-forceMove` flag.

4.6 Advanced Use of SRS Prisma

This section details advanced techniques for using SRS Prisma, and should not be used by novice users.

4.6.1 Split Runs

SRS Prisma has been designed to download and index new data files in the same make process, optimizing use of resources. However, it may be preferable to decouple the download and indexing processes. Therefore, SRS Prisma can be run in two serial processes at different times during a day. Remote updates can be run using:

```
% runPrisma -run <runName> -splitRun download
```

A second run using the same `runName` can then be executed to index any new files without carrying out any additional remote checking:

```
% runPrisma -run <runName> -splitRun index
```

The second run will note any new files downloaded by the first run, and also will block any updates of any libraries where a previous failure has been found (see Section

4.6.2, Run State Preservation, page 145). Reporting will be carried out using the date of the first run, combining information from both stages of the run.

4.6.2 Run State Preservation

The mechanism used in Section 4.6.1, Split Runs, page 144 uses run state preservation to read the state of previous runs. This mechanism can be used independently so that in situations where failures are very common (e.g. overloaded hardware or poor network connectivity). To use run state preservation, use the `-saveState` flag:

```
% runPrisma -run <runName> -saveState
```

Use of this flag forces the flags directory to be read before being cleared, and details of failures and completed offline data sets are preserved. Failed libraries are not updated unless new remote files need to be downloaded, in which case the failure is reset. This mechanism is designed to prevent incorrect offline data being re-indexed. Completed offline sets will be moved online unless data sets need updating.



Note: Run state preservation uses the previous run of the same name e.g. when `-run automatic` is used. Therefore, it should not be used if there have been intervening updates using another run.

CHAPTER

5

SRS PRISMA UPDATE REPORTS

5.1 Introduction

SRS Prisma carries out a complex array of interdependent tasks to update an SRS installation; and, although it may often require little or no SRS administrator attention, SRS Prisma generates a range of in depth but intuitive HTML-based reports, which can help the administrator track down problems and monitor resource usage.

5.2 Creating Reports

SRS Prisma automatically generates a series of HTML reports during the update process executed by `launchPrisma` or `runPrisma`. After each library level has been updated, a brief one page summary known as a 'snapshot' is generated, and a complete, exhaustive report is generated after the entire process has completed. However, it is also possible to generate reports for any SRS Prisma run at any stage in the update in order to assess the progress of a run.

5.2.1 Types of Report

Three different types of SRS Prisma reports can be generated:

- Snapshots
- Full reports with dependency images
- Full reports without dependency images

A snapshot is a single page report designed to show at a glance the status of an update, but without fine detail on individual targets. Alternatively, a full report can be generated, with an HTML report page for each individual target, and for each update stage of each library. The full report usually includes a pre-generated dependency tree diagram (see Section 5.3.13, The Dependency Tree, page 176). However, this can be time consuming and can be turned off when creating reports.

5.2.2 Command Line Arguments

These three types of reports can be generated using the SRS Prisma script `reportPrisma`. Command line arguments can be used to control the type of report produced, the libraries checked and the run reported:

Table 5.1 `reportPrisma` Command Line Arguments

Flag	Arguments	Description
<code>-report</code>	<code>full</code> , <code>noImages</code> or <code>snapShot</code> (default <code>noImages</code>)	Type of report produced
<code>-noTargets</code>	Boolean	Suppress rendering of individual targets as HTML
<code>-run</code>	String (default <code>manual</code>)	Produce report for specified run e.g. <code>manual</code> , <code>automatic</code>
<code>-l</code>	String	List of libraries to report on
<code>-L</code>	String	List of libraries to exclude from report
<code>-g</code>	String	List of groups to report on
<code>-G</code>	String	List of groups to exclude from report
<code>-v</code>	Boolean	Print verbose output
<code>-reportDir</code>	String	Output directory for reports
<code>-help</code>	Boolean	Display options

5.2.3 Methods for Producing Reports


Production of full reports can be time consuming (as much as 30 minutes for a very large run). To produce reports more rapidly, the following methods are suggested:

- Production of rapid, single page reports with `-report snapShot`
- Suppress production of images by using `-report noImages`
- Suppress rendering of individual targets by using `-noTargets`
- Reduce scope of reporting to a few libraries with `-l`, `-L`, `-g`, and `-G`

5.3 Viewing Reports

5.3.1 SRS Prisma Status Color Key

Throughout SRS Prisma Report pages, a common set of colors and icons are used to denote the status of update stages and targets. Figure 5.1 shows the page that summarizes the different status colors and icons:



Color	Icon	State	Description
Green	✓	Completed	Target/stage completed with no failures or errors
Magenta	✓	Completed with Errors	Target/stage completed with no failures but with non-fatal errors
Red	✗	Completed	Target/stage failed to complete
Grey	↶	Skipped	Target/stage skipped due to previous errors
Cyan	⌚	Running	Target/stage started but not yet complete
Purple	⌚	Waiting	Target/stage has yet to be executed
Dark Purple	⌚	Not scheduled	Stage has not been scheduled to run

Figure 5.1 SRS Prisma Color Key

This page is available via the **View SRS Prisma Color Key** at the top of the main SRS Prisma report page.

5.3.2 Getting Help

Each SRS Prisma report page contains a link to the appropriate section of the SRS Help Center in the banner, as shown in Figure 5.2.



Figure 5.2 Help Center Link

Click this link to find out more about different pages and different types of reports.

5.3.3 The Calendar Page

Once generated, either manually or automatically, SRS Prisma reports can be found in the directory structure pointed to by the global parameter `reportDir`. The most recent reports can be found in the subdirectory `current`, but if SRS Prisma is run as an 'automatic' run using `launchPrisma`, the reports are also archived into subdirectories identified by the day of the month (1, 15, 28 and so on) once the run is complete. These directories are rewritten on a rolling monthly basis.

The Calendar Page (`index.html` in this directory) can be used to navigate to both current and archived reports for both Update and Quality reports alike. After SRS Prisma has been installed, but before a successful SRS Prisma run has been completed, the page shown in Figure 5.3 will appear:

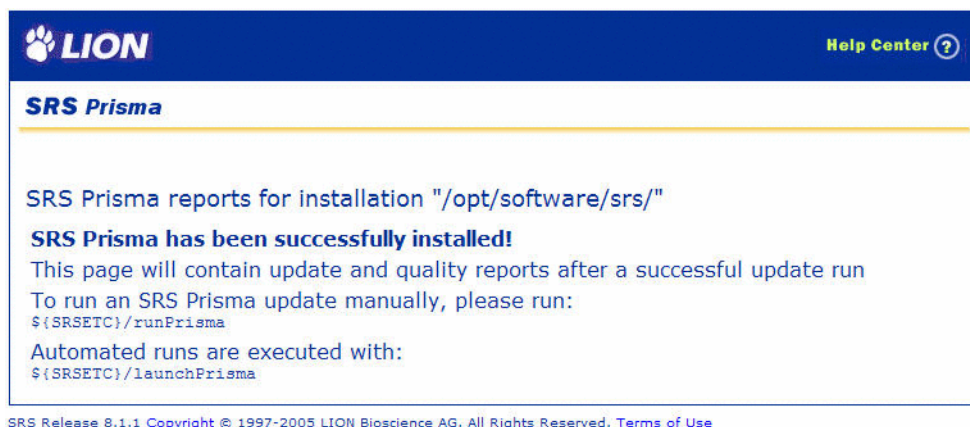


Figure 5.3 Successful Installation Message Page

Once the SRS Prisma update process has been run successfully, a page similar to that shown in Figure 5.4 will be available:

Report Options

SRS Prisma generates 3 different types of reports:

- Update Reports
- Configuration Update Reports
- Quality Reports

Use the radio buttons to choose the report types you want to view

☒ View update report

☐ View configuration update report

☐ View quality report

View Latest Report

Recent Reports

May 2005

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

April 2005

Sun	Mon	Tue	Wed	Thu	Fri	Sat
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

March 2005

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

SRS Release 8.1.1 Copyright © 1997-2005 LION Bioscience AG. All Rights Reserved. [Terms of Use](#)

Figure 5.4 Calendar Page

5.3.4 Update Report Page

To view Update Reports, on the Calendar Page (Figure 5.4) select the option button in the **Report Options** box to **View update report**. To view the current update report, click **View Latest Report**. To view an archived report, click on the number of the day you are interested in from the **Recent Reports** area.

After choosing to see any update report, archived or current, the main Update report page will appear. This has been designed to give a broad overview of the selected SRS Prisma run and to give information on the status of individual libraries scheduled for updating.

If the SRS Prisma run is still in progress and no report has been produced, the page shown in Figure 5.5 will appear:



Figure 5.5 Update Process Running Page

If an SRS Prisma report is still being produced, the page shown in Figure 5.6 will appear.



Figure 5.6 Report Process Running Page

Once a valid report is available, a page similar to the one shown in Figure 5.7 will appear:

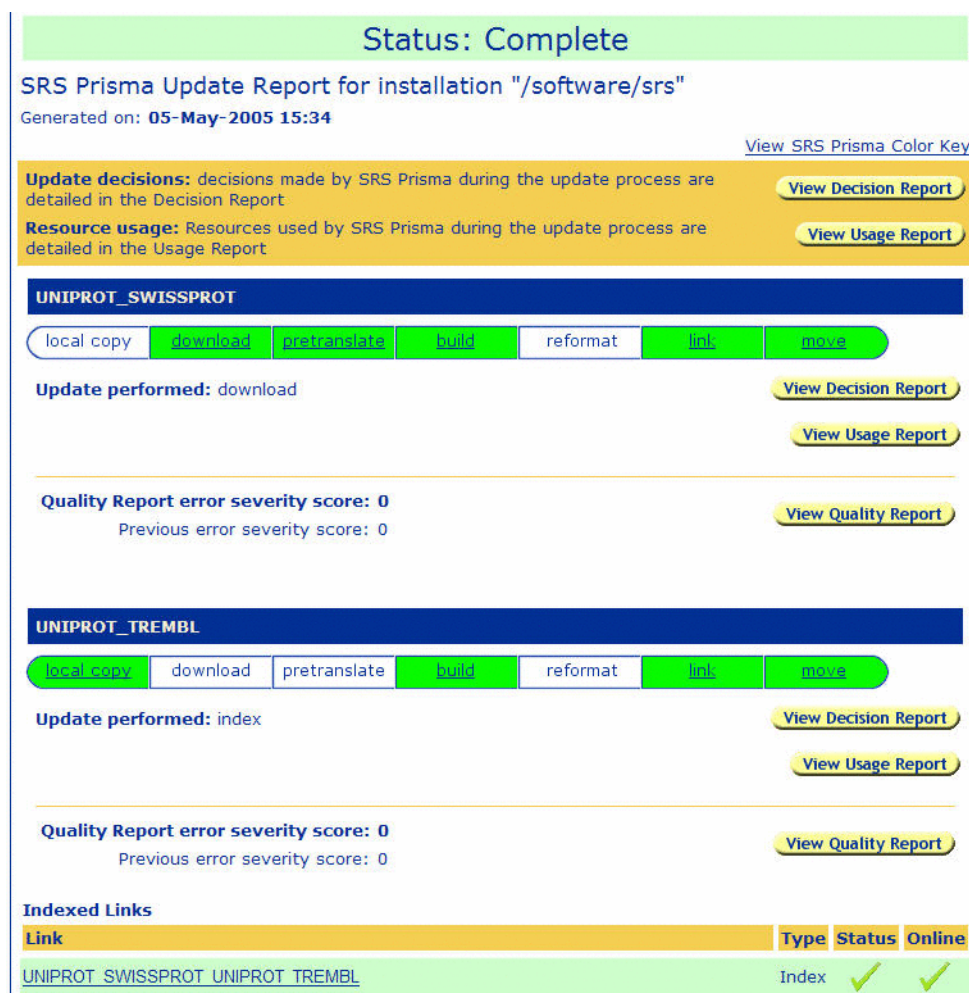


Figure 5.7 SRS Prisma Update Report

This report shows the outcome of a successfully completed update of two databases, and the link between them. The page is divided into three main sections. The first provides general information about the SRS Prisma report, such as the installation involved, whether the run has completed, the type of report and when the report was generated. It also provides a button to navigate to the 'Decision Report' (see Section 5.3.14, The Decision Report, page 178).

The next major section is a list of short graphical summaries for each library updated. Each summary contains the name and type of the library, a graphic representing the update status of the library, the type of update carried out plus a link to the Decision Report for that library and, if available, a summary of the Quality Report findings and a link to the quality Report for that library.

The graphic shows at a glance the status of the library update. The graphic is divided into seven sections, each representing a different stage in the update:

- Local Copy Stage - up-to-date offline files are copied offline
- Download Stage - new files are downloaded from a remote site
- Pretranslate Stage - `unpackCommands` are run on library
- Build Stage - SRS indexing occurs
- Reformat Stage - BLAST/FASTA files produced, `reformatCommand` run on library
- Link Stage - out-of-date links are rebuilt
- Move Stage - completed offline indices and data are moved online

The sections are color-coded according to their status (see Section 5.3.1, SRS Prisma Status Color Key, page 150), which is also shown when the section is pointed to by the mouse. If the report is a full report, the each stage can also be clicked to go to a more detailed stage report. If the report is a snapshot report then these individual pages are not available.

The Decision section indicates the type of update that SRS Prisma has scheduled for that library. The View Decision Report button takes the user to the Decision Report page for the update. Decisions are discussed in more detail in Section 5.3.14, The Decision Report, page 178.

The last section of the page is a table of indexed links that have been scheduled for update. The **Link** column contains the name of the link ('from' library and 'to' library separated by an underscore). In full reports, this is a link to the update report for that link. The **Status** column contains an icon showing the status of the indexing job for that link. The **Online** column contains an icon showing the status of the installation process for that link.

5.3.4.1 Examples of Updated Library Graphics

Figure 5.8 shows an example where action has been scheduled for **download**, **unpack**, **build**, **link** and **move**. **local copy** and **reformat** are white, indicating that the stage is not applicable or no update is necessary. All scheduled stages have

been run to completion and the library has been installed online, and are colored green to indicate success.

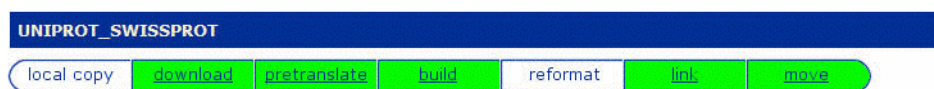


Figure 5.8 A successful library update

Figure 5.9 shows an example where the stages **download**, **build** and **link** are scheduled. **download** is colored green as it has completed successfully. **build** is colored blue as it is still running. **link** is colored purple as it is waiting for **build** to complete before running. There is currently no move scheduled as the outcome of the update is not yet known.



Figure 5.9 A library update that is still running

Figure 5.10 shows an example where the **local_copy**, **download**, **build** and **link** phases are scheduled. **local_copy** is colored green as it has completed successfully. **build** is colored red as it has failed. **link** is colored green as Prisma has still built links using the library, but using the available online indices.

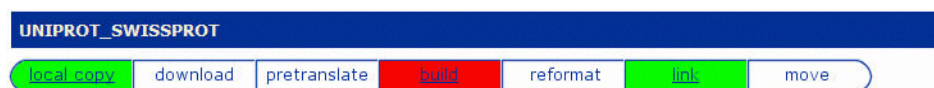


Figure 5.10 A library update which has failed

5.3.4.2 Examples of Updated Link Graphics

The status of SRS indexed links is also shown on the main report page. The name, type, index status and move status of the link is shown.

Figure 5.11 shows an example where the index link from UNIPROT_SWISSPROT to UNIPROT_TREMBL has been successfully created and installed online.

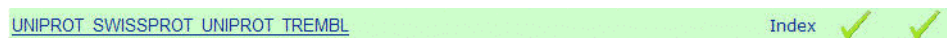


Figure 5.11 A successful link update

Figure 5.12 shows an example where a link that has failed to be built, and hence has not been installed online.



Figure 5.12 A failed link update

Further details on links can be obtained by clicking on the name of the link, which returns a Link Report (see Section 5.3.10, The Link Report, page 168).

5.3.5 Stage Reports

5.3.5.1 General Format of Reports

For full SRS Prisma reports, clicking on individual sections takes the user to a more detailed stage report. All reports follow the same general layout. The top of the page has the name of the stage, its status (color-coded) and, if applicable, a link to the appropriate dependency diagram (see Section 5.3.13, The Dependency Tree, page 176). Next, there is a table with general configuration details and other information pertinent to the stage. Finally, there is a table of the individual processes ('targets') that make up the stage, which are color-coded according to their status. The columns of the target list vary for each type of report, but generally include a target name (linked to the appropriate target report), and an icon indicating the status of the target.

The following sections describe the contents of each type of report in more detail.

5.3.5.2 The Local Copy Report

During the local copy stage, up-to-date datafiles are copied or linked from the online data directory to the offline data directory so that they can be used to build new indices. The stage report shows whether this operation has completed successfully. The example shown in Figure 5.13 shows a successfully completed local copy phase.

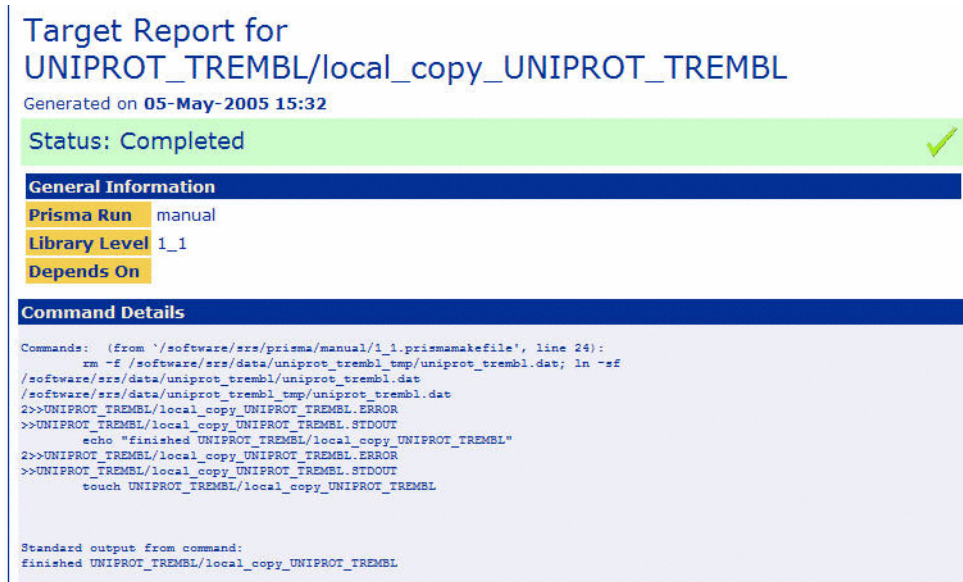


Figure 5.13 Local Copy Report

This report shows the status of the stage, the name of the run and library level, plus the commands run and their output and error (if any).

5.3.5.3 The Download Report

The download stage is responsible for downloading new files from a remote site. This report page provides general information about how the download is configured and what happened during its execution. The following image shows an example for the EMBLNEW database, where one file has been successfully downloaded, but another is still being downloaded:

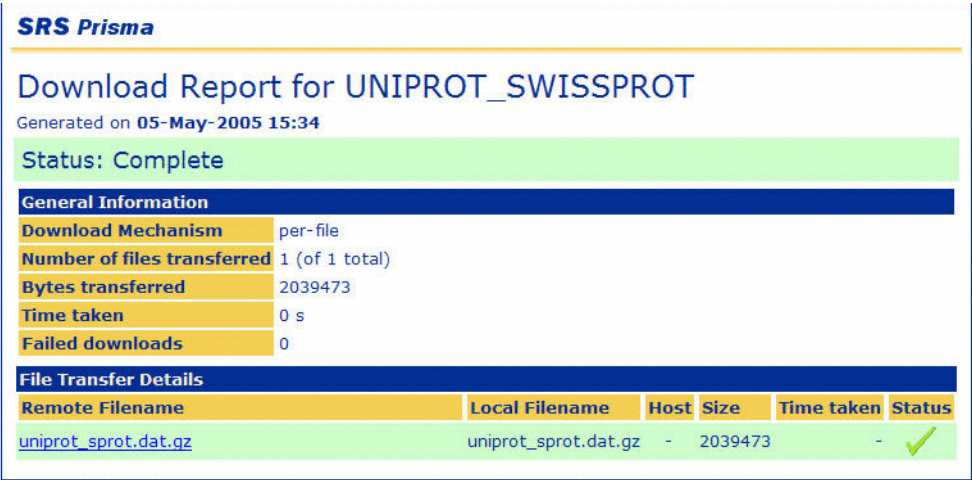


Figure 5.14 Download Report

The following information is available in the **General Information** section:

Download Mechanism

If a library is of parallelType files or fileSize, then downloads are performed as individual processes on each file ('per file'). Otherwise, downloads are performed using a single process ('per database').

Bandwidth limit

The bandwidth per process is limited to this value. 0 (zero) indicates no limit.

Number of files transferred

number of files successfully transferred of the total required.

Bytes transferred

Total file size transferred for all completed processes for this library

Time taken

Time taken for all completed downloads.

Failed downloads

Number of downloads that have not successfully completed.

The rest of the report gives information for each file to be downloaded. The following information is shown (if available):

Remote Filename

Name of file on remote server.

Local Filename

Name of file locally after download and pretranslation.

Host

Name of local host on which execution occurred.

Size

Size of file downloaded in bytes.

Time taken

Time in seconds for the file to download.

Status

The status of the target is denoted by a standard SRS Prisma icon (see Section 5.3.1, SRS Prisma Status Color Key, page 150).

5.3.6 The Pretranslation Report

The pretranslate stage is responsible for carrying out pre-processing commands after new files have been downloaded, but before indexing of those files can start. During the pretranslate stage, the command(s) specified by the `unpackCommand` attribute of the appropriate resource object are run. For instance, compressed downloaded files may need to be unpacked before indexing. This report page provides general information about how the pretranslate stage is configured and what happened during its execution. Figure 5.15 shows an example for the UNIPROT_SWISSPROT database, where a file has been unzipped after being successfully downloaded.



Figure 5.15 Pretranslation Report

The following information is available in the **General Information** section:

Number of commands

Number of separate command strings specified by `$Resource.unpackCommand`.

Number of failures

Number of targets that failed.

Total time elapsed

Time taken for all completed downloads.

The rest of the report gives information for each command in `unpackCommand`. The command is displayed along with its type and status, and with individual reports for each target that make up the command. The command may be 'per-file', where one target exists for each file that has been downloaded (as in Figure 5.15 above), or 'per database', where a single target executes the command once after all files have been successfully downloaded.



Note: Where 'per-file' commands only exist for a library which is of `parallelType` 'files' or 'filesSize' (such as EMBLRELEASE), the download, pretranslate and

build commands for each file are 'chained' together so that these stages can run concurrently, pretranslating and indexing each file as it is downloaded.

The following information is available for each target:

Command

Name of target executed (linked to its detailed target report).

Host

Name of local host on which command was executed.

File

For per-file commands, file for which command is run.

Time taken

Time in seconds for the file to download.

% CPU usage

Maximum CPU usage for the target.

Memory

Maximum memory usage for the target.

Status

The status of the target, denoted by a standard SRS Prisma icon (see Section 5.3.1, SRS Prisma Status Color Key, page 150).

5.3.7 The Build Report

The build stage is the central part of the SRS Prisma update process. During this stage, SRS indices, sort sets and virtual sets are built for each library. The Build Report page shows the processes executed during this phase and their outcome. The example shown in Figure 5.16 is part of a report for the build phase of ENZYME, where the indexing and sorting steps have completed successfully.

Build Report for ENZYME

Generated on 06-May-2005 11:05

Status: Complete

General Information

Parallelisation Mechanism

none

Number of Build Targets

1

Number of Failures

0

Total Elapsed Time

12.500000 s

Build Phase Details

Target Name	Host	Time taken	% CPU usage	Memory	Status
main commands for ENZYME					
ENZYME/ENZYME	-	-	-	-	✓
index commands for ENZYME					
ENZYME/ENZYME_index	watson	12.390000	70	-	✓
sort commands for ENZYME					
ENZYME/ENZYME_sort	watson	0.110000	97	-	✓
ENZYME/ENZYME_sort.cat	-	-	-	-	✓
ENZYME/ENZYME_sort.cat.indexFields_cat	-	-	-	-	✓
ENZYME/ENZYME_sort.cof	-	-	-	-	✓
ENZYME/ENZYME_sort.cof.indexFields_cof	-	-	-	-	✓
ENZYME/ENZYME_sort.des	-	-	-	-	✓
ENZYME/ENZYME_sort.des.indexFields_des	-	-	-	-	✓

Figure 5.16 Build Report Page

This stage is also applicable for virtual libraries like UNIPROT, for which the report shown in Figure 5.17 was produced.

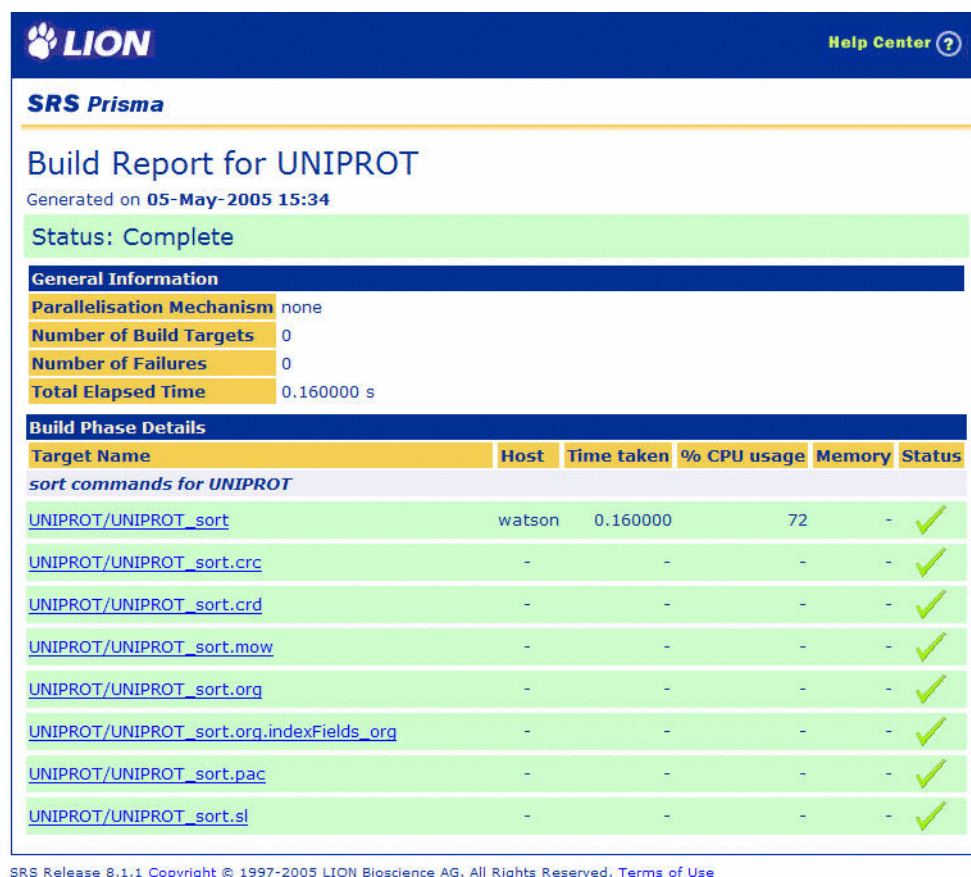


Figure 5.17 Build Report Page for a Virtual Library

The **General Information** section provides the following types of information:

Parallelization Mechanism

Type of parallelization used (`Library.parallelType`).

Part Size

If applicable, maximum size of file parts to index with one process (`Library.partSizeKb`)

Number of Parts

If applicable, number of file parts to index for data file (`Library.partN`)

Number of Failures

Number of failed targets for stage.

Total Elapsed Time

Total time taken so far for all completed targets.

The target list below can be divided into these three categories:

index commands

Targets for which the data files are indexed by `srsbuild`. There may be multiple parts if parallelization is applicable. For virtual libraries, these commands will generate the virtual library objects used during querying.

merge commands

Targets during which partial indices for each field are merges (not applicable if `parallelType` is set to 'none').

sort commands

Targets during which the sort sets used by querying are generated for each field set to sortable.

For each target, the following information may be available:

Target

Name of target executed (linked to its detailed target report).

Host

Name of local host on which the target was executed.

Time taken

Time in seconds for the main command of the target to complete.

% CPU usage

Maximum CPU usage for the target.

Memory

Maximum memory usage for the target.

Status

The status of the target, denoted by a standard SRS Prisma icon (see Section 5.3.1, SRS Prisma Status Color Key, page 150).

5.3.8 The Reformat Report

Figure 5.18 shows a typical Reformat Report page, with each reformat command listed.

Reformat Report for ENZYME

Generated on 06-May-2005 13:25

Status: Complete

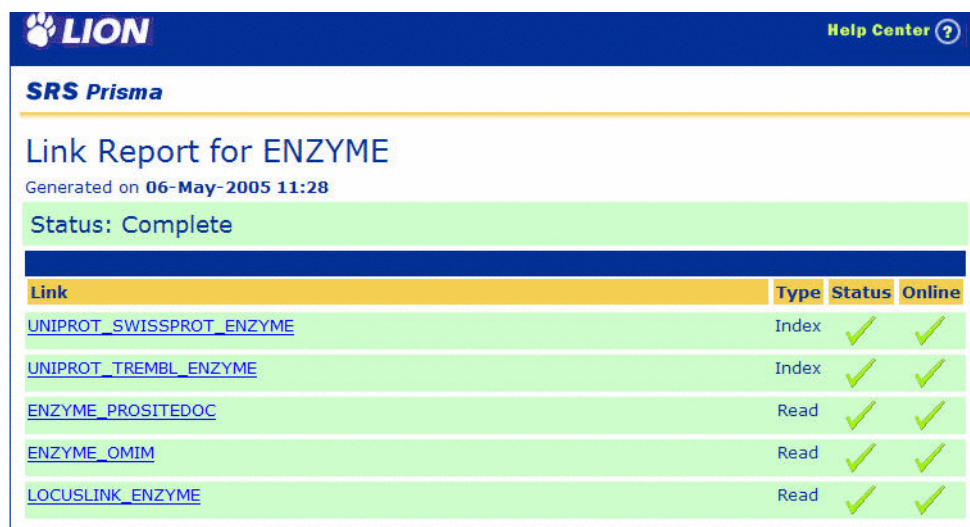
General Information						
Number of Commands	1					
Number of Failures	0					
Total Elapsed Time	0 s					

Command Details						
Command	Host	File	Time taken	% CPU usage	Memory	Status
extractEnzymeNames.sh -inFile %s	-	Per-file	-	-	-	✓
ENZYME/1.reformat_ENZYME_enzyme.dat	-	enzyme.dat	-	-	-	✓

Figure 5.18 Typical Reformat Report

5.3.9 The Link Stage Report

During the link stage, the links in which the library is involved (as either a 'to' or 'from' library) are built. The link stage report lists the relevant links that need building, and indicate their status. The example shown in Figure 5.19 shows the Link Report for ENZYME:



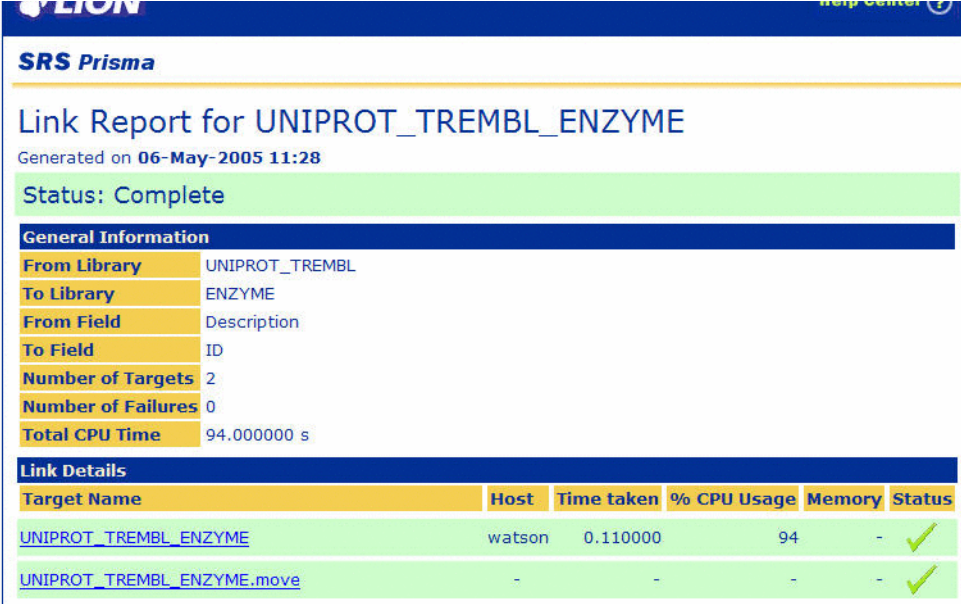
SRS Prisma			
Link Report for ENZYME			
Generated on 06-May-2005 11:28			
Status: Complete			
Link	Type	Status	Online
UNIPROT_SWISSPROT_ENZYME	Index	✓	✓
UNIPROT_TREMBL_ENZYME	Index	✓	✓
ENZYME_PROSITEDOC	Read	✓	✓
ENZYME_OMIM	Read	✓	✓
LOCUSLINK_ENZYME	Read	✓	✓

Figure 5.19 Typical Link Stage Report

Each link is shown with its name, which acts as a link to the detailed report for that link, plus icons showing the status of the link and whether it has been installed online. In this instance, all links have been successfully completed, apart from that from UNISEQ (which has failed) and that from UNIEST (which has not yet been built). None of the links have been installed online.

5.3.10 The Link Report

The Link Report displays the status of all the activities involved in building a link between two libraries, including building and merging indices, and installing the link online. Figure 5.20 shows a successfully completed index link between UNIPROT_TREMBL and ENZYME:



SRS Prisma

Link Report for UNIPROT_TREMBL_ENZYME

Generated on 06-May-2005 11:28

Status: Complete

General Information	
From Library	UNIPROT_TREMBL
To Library	ENZYME
From Field	Description
To Field	ID
Number of Targets	2
Number of Failures	0
Total CPU Time	94.000000 s

Link Details					
Target Name	Host	Time taken	% CPU Usage	Memory	Status
UNIPROT_TREMBL_ENZYME	watson	0.110000	94	-	✓
UNIPROT_TREMBL_ENZYME.move	-	-	-	-	✓

Figure 5.20 Typical index link report

Figure 5.21 shows the status of the read link between UNIPROT_SWISSPROT and PDB, which has completed:

SRS Prisma

Link Report for UNIPROT_SWISSPROT_PDB

Generated on 06-May-2005 11:28

Status: Complete

General Information

From Library	UNIPROT_SWISSPROT
To Library	PDB
From Token	readlink pdbR
To Field	ID
Parallel Type	fileSize
Part Size	100000 kb
Number of Targets	3
Number of Failures	0
Total CPU Time	28.000000 s

Link Details

Target Name	Host	Time taken	% CPU Usage	Memory	Status
UNIPROT_SWISSPROT/1.UNIPROT_SWISSPROT_readlinks_part	-	-	-	-	✓
UNIPROT_SWISSPROT/UNIPROT_SWISSPROT_readlinks	-	-	-	-	✓
UNIPROT_SWISSPROT_PDB	watson	0.370000	28	-	✓

Figure 5.21 Typical read link report



Note: The build targets for this link may include building other read links from UNIPROT_SWISSPROT which are built in the same process.

For each Link Report, the following general information is shown (where available):

From Library

Library from which link is defined.

To Library

Library to which link is defined.

From Field

Index links only, field in from library to use for link.

From Token

Read links only, token of from library to use for link.

To Field

Field in to library to use for link.

Parallel Type

Read links only, parallel type of from library.

Part size

Read links only and where applicable, maximum size of file parts to use for building.

Number of parts

Where applicable, parts to build link in.

Number of Failures

Number of failed targets for link.

Total CPU Time

Total time taken for completed targets.

For each target, the following information is shown (where available):

Target name

Name of target (also link to detailed target report).

Host

Where applicable, execution host for target.

Time taken

Where applicable, time elapsed for completion of target

Memory

Where applicable, maximum memory usage of target.

%CPU Usage

Where applicable, maximum CPU usage of target

Status

Icon showing status of target.

A set of read links may contain links to libraries built at different stages of the update process. For instance, read links are built from UNIPROT_SWISSPROT to GENBANKRELEASE (updated directly) and GENBANKNEW (updated as a dependent library). In this case, there may be multiple read link build targets where different commands have been run. These are distinguished with a suffix that shows

the stage (e.g. 1.UNIPROT_SWISSPROT_readindex_2). The same is also true where a read link to or from a 'blocked' library has been rebuilt, where the suffix _rebuild indicates the target (e.g. 1.UNIPROT_SWISSPROT_readindex_rebuild).

5.3.11 The Move Report

After all updating for a library has completed and if it was successfully updated, it is installed online. The indices (if the library has them) and data files are moved online by a series of commands. The Move Report page shows the status of the move commands specified. Figure 5.22 shows a portion of the move report for the ENZYME database:

Generated on 06-May-2005 11:28	
Status: Complete	
General Information	
Install Type move	
Move Phase Details	
Target Name	Status
move all commands for ENZYME	
ENZYME/ENZYME move all	✓
move data commands for ENZYME	
ENZYME/1.ENZYME move data	✓
move indices commands for ENZYME	
ENZYME/1.ENZYME move indices	✓
ENZYME/10.ENZYME move indices	✓
ENZYME/11.ENZYME move indices	✓
ENZYME/12.ENZYME move indices	✓
ENZYME/13.ENZYME move indices	✓
ENZYME/14.ENZYME move indices	✓
ENZYME/15.ENZYME move indices	✓

Figure 5.22 Move Report

The **General Information** section contains the following information (where applicable):

Install Type

Type of installation to use, such as move, switchlink, command, none (`Resource.installType`).

Install Files

If specified, additional files to move online (`Resource.installFiles`).

Install Command

If specified, command to use for installation instead of standard move (`Resource.installCommand`).

The list of targets is divided into the following sections (where appropriate):

move all commands

Commands to finalize the install process.

move data commands

Commands to install data files online and remove/archive unused offline/online data.

move indices commands

Commands to install SRS indices and sort/virtual sets online.

Each target in each phase is listed with the following information:

Target name

Name of move target (linked to a detailed target report).

Status

Status of target.

5.3.12 The Target Report

In addition to the general stage reports, SRS Prisma also provides individual reports on each target executed during the update process. These reports provide detailed information on the commands executed, process timing and standard output/error. A target report can be viewed by clicking on the name of the target in the appropriate stage report. Figure 5.23 shows a portion of the target report for 1.UNIPROT_SWISSPROT_part, a partial indexing target for UNIPROT_SWISSPROT.

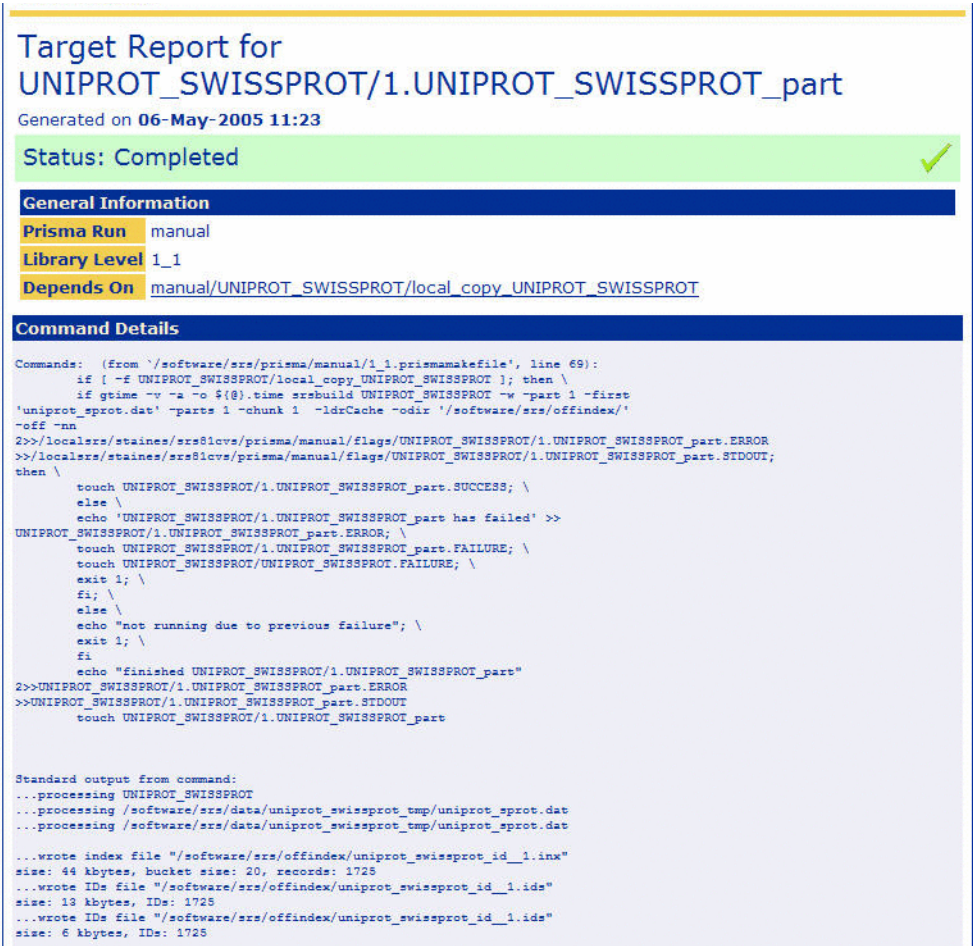


Figure 5.23 Typical Target Report

The following general information is available for all targets:

- SRS Prisma Run**
Run name, such as manual or automatic.
- Library Level**
Library level and round number.
- Depends On**
Link to report for target on which this target depends (may be empty).

In addition, for most targets the following detailed information is available:

Execution host

Local host on which process was executed.

Start Time

Time at which process started.

Elapsed Time

Total time taken for process to run.

System Time

System time spent by process.

Max Memory

Maximum memory used by process.

% CPU Usage

Maximum CPU usage.

Exit Status

Exit status code returned by command.

This information may not be present for all targets, for example, move targets do not have associated process information as shown in Figure 5.24.

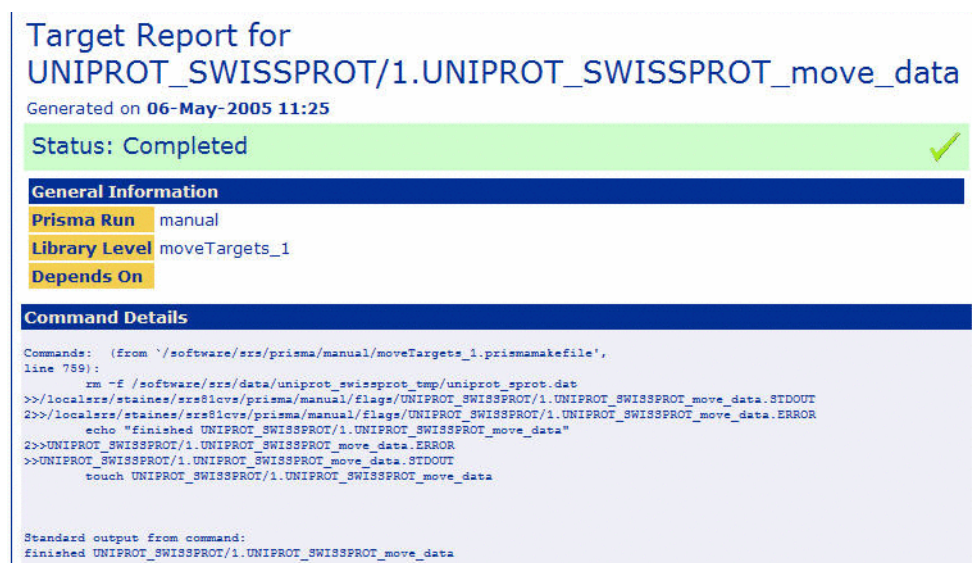


Figure 5.24 Target Report for Move Targets

5.3.13 The Dependency Tree

In addition to the text-based HTML reports generated by SRS Prisma, a dependency tree diagram can also be produced for each report. This allows the administrator to see which jobs depend on each other, and gives an overview of the update process. In addition, timing information is also displayed so the resource usage for an update can be assessed for possible bottlenecks.

By default, dependency trees are not produced as they are time-consuming. For more information about activating these reports, please consult Section 3.6.3, Generate Dependency Tree Images, page 49.

Where available, the Dependency Tree for each stage of each library (barring move and link stages) can be viewed by clicking the **View Dependency Tree** link at the top of each stage Report.

Figure 5.25 shows the Dependency Tree for the download stage of TAXONOMY:

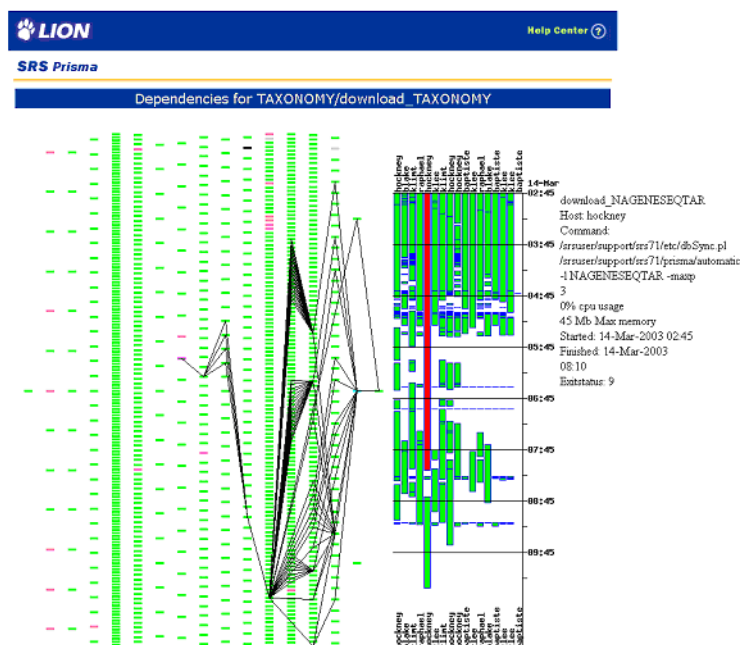


Figure 5.25 Dependency Tree

The report is divided into two sections. The image on the left is a dependency tree for the selected stage, showing the chain of dependent and required targets for selected targets. The targets are represented by color-coded rectangles showing their status, using the standard SRS Prisma color scheme. These rectangles can be clicked to produce an individual target report, or moused-over to provide timing information, and to highlight the corresponding process in the right-hand chart. The currently selected rectangle (`download_TAXONOMY`) is outlined in magenta.

The image on the right is a chart of process timings. Each vertical bar shows the time taken for each update process, color-coded to show status. The different columns reflect different numbers of processors available on different hosts. This chart can be used to identify time-consuming single processes that might be parallelized. Again, each rectangle can be clicked to obtain a detailed target report, or moused-over to highlight the target in the left-hand diagram and show timing information.

In the example shown in Figure 5.25, the long red rectangle on the timing chart has been moused-over to identify it as `download_NAGENESEQTAR`, and the corresponding rectangle in the chart has been highlighted in black.

5.3.14 The Decision Report

Although SRS Prisma is designed to be run as an automatic process, and to take all steps needed to bring an installation up-to-date, it is sometimes useful to examine the decisions made by SRS Prisma during the update process. Decision Reports provide this kind of information to the administrator.

Decision information is available to the administrator in two locations. Firstly, the individual library graphics on the main report page have decision information associated with them. Figure 5.26 shows the graphic for the UNIPROT_SWISSPROT library, for which new files need to be downloaded:



Figure 5.26 Decision Information on the Main Report Page

Clicking the **View Decision Report** button for a library will show the appropriate section of the main decision report (also available by clicking **View Decision Report** at the top of the page). Figure 5.27 shows a typical Decision Report page.

Update Decision Report for installation
"/software/srs"

Generated on: **05-May-2005 15:34**

Round 1 of update process

Libraries from update level 1

[View check phase output](#)
[View update phase output](#)
[View update phase errors](#)

UNIPROT_SWISSPROT

Update action: download

[View status of library before update](#) [View status of library after update](#)

Reason: The following new files are on the remote site and need downloading:

- uniprot_sprot.dat.gz

The following links involving this library needed to be built:

- index link from UNIPROT_SWISSPROT to UNIPROT_TREMBL
- virtual link from UNIPROT to UNIPROT_TREMBL via UNIPROT_SWISSPROT
- virtual link from UNIPROT to UNIPROT_SWISSPROT via UNIPROT_TREMBL

UNIPROT_TREMBL

Update action: index

[View status of library before update](#) [View status of library after update](#)

Reason: UNIPROT_TREMBL needs reindexing as no valid online ID index has been found.

The following links involving this library needed to be built:

- index link from UNIPROT_SWISSPROT to UNIPROT_TREMBL
- virtual link from UNIPROT to UNIPROT_TREMBL via UNIPROT_SWISSPROT
- virtual link from UNIPROT to UNIPROT_SWISSPROT via UNIPROT_TREMBL

UNIPROT (virtual library)

Update action: rebuild

[View status of library before update](#) [View status of library after update](#)

Reason: Virtual library 'UNIPROT' needs rebuilding as the following member libraries have been updated:

- UNIPROT_SWISSPROT
- UNIPROT_TREMBL

Figure 5.27 Decision Report Page

The Decision Report is divided into sections, for each set of libraries to be updated such as library level 1 of round 1). In each section, the following information is available:

- check phase output
- update phase output
- update phase errors
- library decisions

The check phase output can be viewed by clicking the appropriate **View check phase output** link for each library level. The check phase output contains the brief summary of the updates required that is produced by the `prismacheck` application. This can be useful for debugging if the `-v` flag was set for `runPrisma`. Figure 5.28 shows typical check phase output.



Figure 5.28 Check Phase Output

The update phase output can be viewed by clicking the appropriate **View update phase output** link for each library level. The update phase output contains the output of the `make` command, including each command that was run. Figure 5.29 shows typical update phase output.

```

if gtime -v -a -o UNIPROT_SWISSPROT/download_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz.time
then \
echo "finished UNIPROT_SWISSPROT/download_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz" 2>>UNIPROT_SWISSPROT/download_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz.SUCCESS; \
else \
echo 'UNIPROT_SWISSPROT/download_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz failed' >>UNIPROT_SWISSPROT/download_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz.FAILURE; \
touch UNIPROT_SWISSPROT/download_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz.FAILURE; \
exit 1; \
fi
touch UNIPROT_SWISSPROT/download_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz; \

if [ ! -f UNIPROT_SWISSPROT/pretrans_UNIPROT_SWISSPROT.STAGEFAILED ]; then \
if gtime -v -a -o UNIPROT_SWISSPROT/1.pretrans_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz.time
then \
echo "finished UNIPROT_SWISSPROT/1.pretrans_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz" 2>>UNIPROT_SWISSPROT/1.pretrans_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz.SUCCESS; \
touch UNIPROT_SWISSPROT/1.pretrans_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz.SUCCESS; \
else \
echo $? > UNIPROT_SWISSPROT/1.pretrans_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz.FAILURE; \
echo 'UNIPROT_SWISSPROT/1.pretrans_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz failed' >>UNIPROT_SWISSPROT/1.pretrans_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz.FAILURE; \
touch UNIPROT_SWISSPROT/pretrans_UNIPROT_SWISSPROT.STAGEFAILED; \
touch UNIPROT_SWISSPROT/1.pretrans_UNIPROT_SWISSPROT_uniprot_sprot.dat.gz.FAILURE; \
fi; \
fi

```

Figure 5.29 Update Phase Output

In addition, the **View update phase output** link also shows a similar page for the standard error from the make command. It is rare that this page contains errors. When it does, the errors are usually caused by formatting problems (such as newlines in `unpackCommand` or `reformatCommand`) or are produced by a batch queuing system (such as Platform LSF).

For each library examined, the type of update required and reasons for this update are shown. Figure 5.30 shows an example where UNIPROT_SWISSPROT needs updating as new remote files have been found.

Update action: download

[View status of library before update](#)

[View status of library after update](#)

Reason: The following new files are on the remote site and need downloading:

- uniprot_sprot.dat.gz

The following links involving this library needed to be built:

- index link from UNIPROT_SWISSPROT to UNIPROT_TREMBL
- virtual link from UNIPROT to UNIPROT_TREMBL via UNIPROT_SWISSPROT
- virtual link from UNIPROT to UNIPROT_SWISSPROT via UNIPROT_TREMBL

Figure 5.30 Update Action

The reasons given may also include why a library was *not* updated. This includes virtual libraries for which the parent libraries failed to be successfully updated.

update

New data files have been found on the remote server, or local online files have been found to be obsolete and not required.

rebuild

Some or all SRS indices need creating for this library. This may be because of local data that is newer than the indices, or missing indices/sort sets.

create

The dependent library or virtual library needs to be updated because one of its parents has been updated.

link

A link to or from this library needs to be rebuilt as its indices are out-of-date or missing.

install

A complete set of data and indices for the library exists offline and needs to be moved online.

none

No action needs to be taken for this library.

In addition, these reports include information on whether libraries were blocked from being moved online due to failure. Figure 5.31 illustrates this with an example where the failure of the virtual library UNIPROT has caused UNIPROT_TREMBL to be blocked from being installed online, with the result that the updated link from UNIPROT_TREMBL and ENZYME has to be rebuilt:

UNIPROT_TREMBL

Update action: download

[View status of library before update](#) [View status of library after update](#)

Reason: The following new files are on the remote site and need downloading:

- uniprot_trembl.dat.gz

The following links involving this library needed to be built:

- index link from UNIPROT_TREMBL to ENZYME
- virtual link from UNIPROT to ENZYME via UNIPROT_TREMBL

Installation blocked

This library has been blocked from installation because the related library UNIPROT has failed

As a consequence, the following links need to be rebuilt using online data

- index link from UNIPROT_TREMBL to ENZYME

Figure 5.31 Decision Report showing a blocked library

For debugging purposes, more information is provided in the form of the Trace Reports. These capture the physical state of the data and indices for each library before and after updating has taken place. The Trace Reports can be obtained by clicking on **View status of library before update** and **View status of library after update** for each library on the Decision Report. Figure 5.32 shows an example of the pre-update Trace Report for UNIPROT_TREMBL:

```
pre-process file report for UNIPROT_TREMBL
Generated on 05-May-2005 15:31

Online file listing for /software/srs/data/uniprot_trembl/:
-rw-r--r-- 1 srsadmin biosys 2409384 Apr 21 19:13 uniprot_trembl.dat

Offline file listing for /software/srs/data/uniprot_trembl_tmp/:
No files found

Online index directory listing
-rw-r--r-- 1 srsadmin biosys 512 May 5 15:18 emblcontigs_uniprot_trembl.ids
-rw-r--r-- 1 srsadmin biosys 94 May 5 15:20 emblcontigs_uniprot_trembl.inx
-rw-r--r-- 1 srsadmin biosys 134 May 5 15:19 embl_emblcontigs_uniprot_trembl.virtualL_.om
-rw-r--r-- 1 srsadmin biosys 134 May 5 15:19 embl_emblnew_uniprot_trembl.virtualL_.om
-rw-r--r-- 1 srsadmin biosys 134 May 5 15:19 embl_emblrelease_uniprot_trembl.virtualL_.om
-rw-r--r-- 1 srsadmin biosys 134 Apr 27 10:27 embl_emblwgs_uniprot_trembl.virtualL_.om
-rw-r--r-- 1 srsadmin biosys 512 May 5 15:18 emblnew_uniprot_trembl.ids
-rw-r--r-- 1 srsadmin biosys 94 May 5 15:20 emblnew_uniprot_trembl.inx
-rw-r--r-- 1 srsadmin biosys 11776 May 5 15:18 emblrelease_uniprot_trembl.ids
-rw-r--r-- 1 srsadmin biosys 40984 May 5 15:20 emblrelease_uniprot_trembl.inx
-rw-r--r-- 1 srsadmin biosys 134 May 5 15:19 emblwgs_emblwgsnew_uniprot_trembl.virtualL_.om
```

Figure 5.32 Trace Report

This indicates the date and size of the data and index files for the online and offline directories for the library, and may assist in determining why an update did or did not occur.

5.3.15 The Usage Report

SRS Prisma also includes reports of changes to file and system usage by libraries during the update process. These are generated automatically at the end of the update process, and if available, can be viewed by clicking on the **View usage report** button on the main report page. The main page of the usage report gives an overview of the resource usage by all libraries during the update process. Each update is represented by a graphic showing the following information:

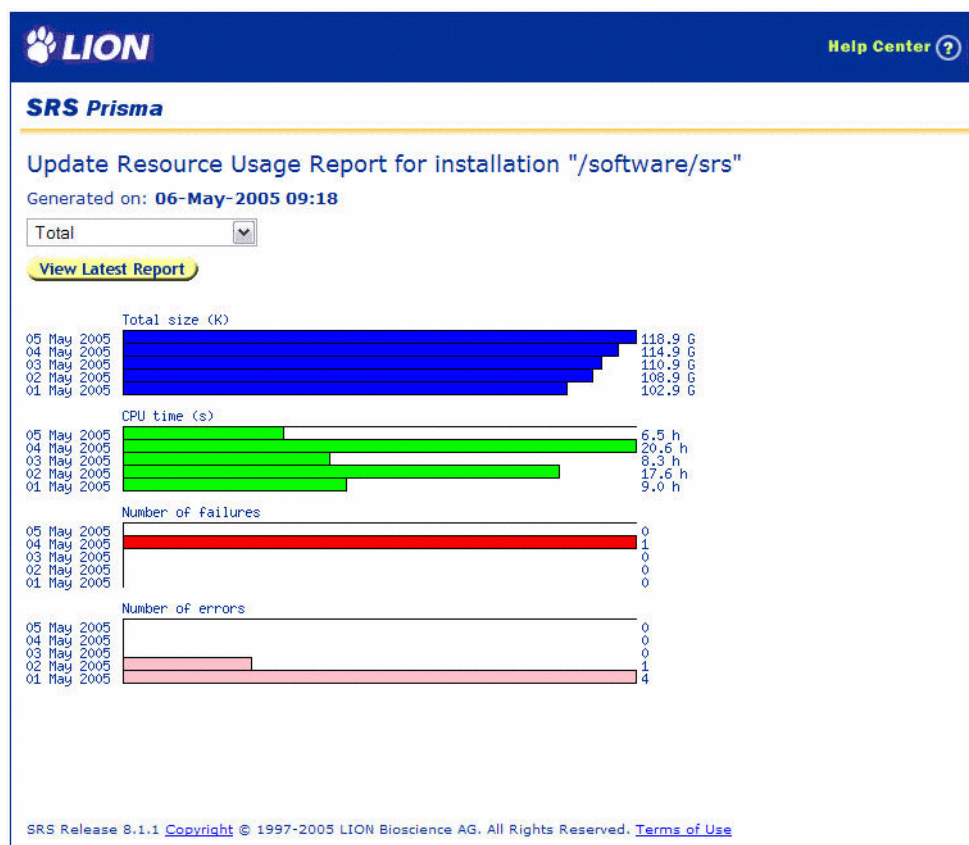


Figure 5.33

- Total time - the total CPU time expended by individual update processes (this does not include additional overhead including reporting etc.)
- Total size - the total size of the data files and indices associated with the libraries after the update (this does not include additional, unindexed files such as FASTA files and BLAST indices).
- Total failures - the total number of failed update processes during the last run
- Total errors - the total number of update processes that produced standard error during the last run.

A graphic with these properties is displayed for each of the last 30 updates, so any trend over time of these properties can be observed.

In addition to this overview, each. To obtain a report on an individual library, select the name from the menu at the top of the main update page and click **View usage report**.

The library update page is similar in form to the main usage report, and shows a graphic for the following properties over 30 days:

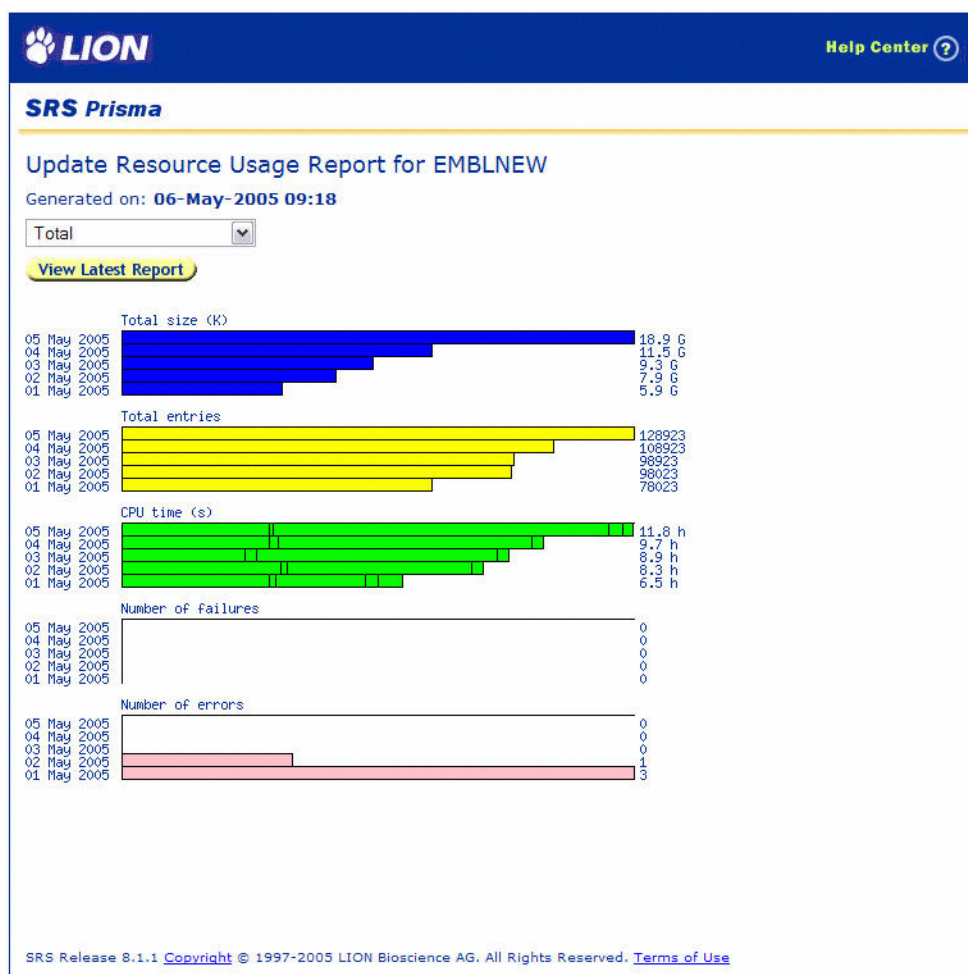


Figure 5.34 The usage report for a library.

- Total time - the total number of seconds of CPU time expended by individual update processes on this library (this does not include additional overhead including reporting etc.)

- Total size - the total size in kb of the data files and indices associated with this library after the update (this does not include additional, unindexed files such as FASTA files and BLAST indices).
- Total entries - the total number of entries in the library.
- Total failures - the total number of failed update processes associated with this library during the last run.
- Total errors - the total number of update processes associated with this library that produced standard error during the last run.

The production of a usage report and the number of days displayed in the total and library graphics can be configured using the Visual Administration tool or directly (described in Section 3.6.4, Number of past runs in usage report, page 49)

5.3.16 The Quality Report

To view Quality Reports, on the Calendar Page (Figure 5.4 on page 153) select the option button in the Report Options box to 'View quality report'. To view the current quality report, click **View Latest Report**. To view an archived report, click on the number of the day of interested in from the calendar area.

At the end of the SRS Prisma update process, the Quality Report module of SRS Prisma is run on all checked libraries. During this process (discussed in depth in Chapter 7, SRS Prisma Quality Report), a score is assigned to each library indicating the numbers and criticality of any configuration problems found. This score, plus the score for the last time the library was checked, is displayed for each library on the main report page (if available), as shown in Figure 5.35.

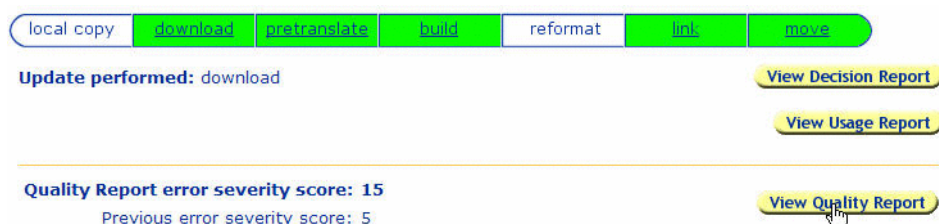


Figure 5.35 Quality Report Information on Main Report Page

This allows the administrator to easily see if problems associated with a library are caused or fixed by obtaining new data or re-indexing existing data. There is also a

button to view the quality report for the library in question (see Chapter 7, SRS Prisma Quality Report).

CHAPTER

6

SRS PRISMA - TROUBLESHOOTING

6.1 Introduction

Although SRS Prisma is normally straight-forward to use and configure, complex configurations can occasionally be difficult to trouble-shoot. With this in mind, this chapter describes a few key concepts and some examples of typical problems. However, if problems persist, members of the SRS Support team are always glad to help and advise on SRS Prisma configuration problems. The process is accelerated if the steps in this chapter have been followed to try to isolate the problems. The .it and .i files should be supplied for the relevant database library. It can also be useful to send a tarball containing the relevant `$SRSPRISMA` subdirectory and `$SRSSWWW/prisma` report directory. It may also be useful to check the errors described in Chapter 10, SRS Prisma - Common Errors

6.2 Finding Errors

The main task that confronts the Administrator when a SRS Prisma run fails to complete fully is to locate the source of the problem. This is generally accomplished using the SRS Prisma reports to locate stages that have either failed (or are still running). Failed targets can then be found in the list of targets for that stage, and the Target Report examined. This generally provides all required information on runtime failures. If a library has been updated when not expected (or not updated when expected), the **Decision Report** page shows the reasoning used by SRS Prisma during the check process.



Note: Experienced Administrators may also wish to examine the contents of the flags directory (`$SRSPRISMA/<run>/flags`) for more details.

SRS Prisma 4.1.1 provides extensive logging of the check phases, by writing full verbose and diagnostic logs to `$SRSPRISMA/DATE/flags/<level>_<round>.prismacheck.log` where `level` and `round` represent the stage of the Prisma run. These logs can be very useful in determining why a particular course of action was taken.

6.3 Isolating Errors

If it is not immediately clear why a particular error is happening, then the next step is to try to isolate the problem to a particular library, and a particular stage in that library. It is advisable to try updating a small test library independently, ensuring that each stage works correctly before attempting to test its behavior with other libraries to which it may be linked. It is also worthwhile setting up a small, local test FTP or HTTP server with small files to debug configurations for complex dependencies.

6.4 Manually Tracking Datafiles

When trying to isolate a problem, it is useful to determine which files are involved, because mismatches between file locations and processes that depend on those files are frequent causes of problems.

The Decision Report can show the administrator the names of any new files to be downloaded, offline files that need installing or obsolete files that need removing. This information, used in conjunction with the Trace Reports for a library, can be extremely useful in tracking changes.

In addition, experienced SRS administrators may find that looking in the `$SRSPRISMA/<run>/flags/<DBNAME>` directory can often help find out which files are involved. `<DBNAME>.new`, `<DBNAME>.removed`, `<DBNAME>.updated` and `<DBNAME>.okay` are lists of filenames for a given database that are to be downloaded, will be removed in the move phase, are already present offline, or are up-to-date and online respectively. In each file, the local and remote name of each file is given. `<DBNAME>.remote` contains a list of all files on the remote site, and `<DBNAME>.transferred` contains a list of all files that have been transferred during the update process. These files are found in the subdirectory for each process.

Once the files involved have been determined, the online and offline data directories can be checked to see if the files have been downloaded, symbolically linked from the online directory or unpacked, and whether those present match the definition of `$Library.searchName` or `$Library.files`. If the failure occurs at the unpack or reformat stage, it is worth checking to see whether the commands specified in the appropriate `prismamakefile` act on the files present, and whether the specified files will always be present.

In addition to the reformat and unpack phases, problems frequently occur during the move phase. In this case, the `<DBNAME>_move_data.makefile` and `<DBNAME>_move_indices.makefile` contain lists of files to be moved online. This should be checked against files downloaded, generated and normally present. For instance, a common mistake is when `installFiles` is used to specify additional files to move online, but those files are not necessarily downloaded or created during each SRS Prisma run.

6.5 Correcting Errors

Depending on the type of error, there are a number of different ways in which SRS Prisma can be rerun to recover from errors. For errors like poor network connectivity or memory/disk space shortage, the `restartPrisma` command can be used to rerun any failed targets (see Section 4.4.2, Restarting SRS Prisma, page 141).

However, in general, if there are errors in configuring individual library `$Resource` class objects, it is usually best to rerun SRS Prisma from scratch, since names of existing files and so on, may be incorrect.

CHAPTER

7

SRS PRISMA QUALITY REPORT

7.1 What is SRS Prisma Quality Report?

SRS Prisma Quality Report is a tool for administrators which they can use to check an SRS installation for configuration problems that should be fixed. With SRS Prisma Quality Report, the day-to-day job of keeping an SRS server running smoothly should become easier.

Included in SRS Prisma Quality Report are checks for:

- Incorrect configuration of SRS
- Integrity of library indices, including link indices
- Parser reliability
- Basic relational database set-up (if SRS Relational has been installed)
- Tools integration

SRS Prisma Quality Report generates a graphical HTML report that shows where errors in an SRS installation are. The report details what errors have been found, for which databases they were found, and suggests how the problem should be fixed. Historical information is stored by SRS Prisma Quality Report and charted such that an administrator can easily pinpoint when a problem has arisen. When serious errors are detected, SRS Prisma Quality Report will e-mail the SRS administrator with a report of what should be fixed immediately to guarantee error-free operation.

SRS Prisma Quality Report is not heavily CPU intensive and on a medium-sized installation, the error checking and compilation activities should be completed on a single processor in approximately 20-30 minutes. While the error checking and report generation activities of SRS Prisma Quality Report are taking place in the background, the SRS installation server should not be affected. Therefore, you can concurrently query the SRS installation as usual.

7.2 Tests in Prisma Quality Report

Quality Report currently tests for more than thirty types of errors that might affect the smooth running of an SRS installation. Most of the routine errors checking activities are designed based on the most commonly encountered problems an SRS administrator might face. Through user feedback, we aim to design a collection of error testing activities as extensive as possible. Therefore, if the user finds any type of error that warrants special attention but that is not covered by the current Quality Report, e-mail support@uk.lionbioscience.com.

7.3 Running SRS Prisma Quality Report

SRS Prisma Quality Report can be run in the following ways:

- As part of an SRS Prisma run.
- Launched stand-alone from the command line.

7.3.1 Running SRS Prisma Quality Report with Prisma

Once Prisma has completed updating of databases, Prisma will initiate Quality Report to start checking the integrity of your SRS installation. Subsequently, Quality Report for each databank on your SRS installation can be viewed from the Prisma Update Report page, to reflect any changes in the status of your SRS installation after the current round of indexing on new data. This is especially useful in terms of checking parser reliability after data update, as from time to time; the format of data might change with new release.

7.3.2 Running SRS Prisma Quality Report Stand-Alone

Alternatively, Quality Report could be launched on its own from the command line. It is useful as a diagnostic housekeeping tool to monitor changes in SRS installation brought about by day-to-day maintenance alterations or a customization project.



Note: Reports generated from Quality Reports launched on command line will not be archived. Launching Quality Report on command line mainly serves to provide a convenient snapshot of the overall status of your SRS server at any time point.

As archiving of Quality Report is synchronized with Update Report in a complete Prisma run, the reports generated will only be archived when it is executed as part of a Prisma run. If the user wants to keep track of changes of results in Quality Report, running Quality Report as part of a complete Prisma is strongly recommended.

7.3.3 Running SRS Prisma Quality Report from the Commandline

SRS Prisma Quality Report is executed by running the `srsinspect icarus` script from the commandline with:

```
srsinspect
```

This will commence the error checking activities in SRS Prisma Quality Report, and compile the errors found in a graphical HTML report once all the error-checking activities are completed.

Option Flags

A collection of command line option flags is available for use with the `srsinspect` command. On the commandline, you can run SRS Prisma Quality Report with the optional utilities set to your requirements.

When you run SRS Prisma Quality Report at the command line, you can use the following optional flags to fine-tune the error checking activities:

Table 7.1 Commandline option flags

Flag	Parameter Name	Type	Default Value	Description
-l	Databanks	String	"	To run tests on the databanks listed within this list of databanks. For example, <code>srsinspect -l 'gen-peptnew genbanknew'</code> will tell SRS Prisma Quality Report to run tests only on <code>genpeptnew</code> and <code>genbanknew</code> .
-L	Exclude the databanks	String	"	To run tests on the databanks listed within this list of databanks. For example, <code>srsinspect -L 'gen-peptnew genbanknew'</code> will tell SRS Prisma Quality Report to run tests only on all databanks excluding <code>genpeptnew</code> and <code>genbanknew</code> .

Table 7.1 Commandline option flags

Flag	Parameter Name	Type	Default Value	Description
-g	Groups	String	"	<p>To run tests on the databanks listed within the group. You can use either the name of a group or the short name of a group for this purpose.</p> <p>For example, <code>srsinspect -g SSQ</code> will tell SRS Prisma Quality Report to run tests on all databanks in the group of sequence sub-sections. (group shortname <code>SSQ</code>).</p>
-G	Exclude the groups	String	"	<p>To run tests on the databanks listed within the group. You can use either the name of a group or the shortname of a group (if available) for this purpose.</p> <p>For example, <code>srsinspect -G SSQ</code> will tell SRS Prisma Quality Report to run tests on all databanks BUT NOT those listed in the group of sequence sub-sections. (group shortname <code>SSQ</code>).</p>
-n	Number of test enteries per databank checked	Int	20	<p>To run tests on at least n test entries in each of the databanks</p> <p>For example, <code>srsinspect -n 5</code> will tell SRS Prisma Quality Report to run tests on at least 5 entries picked from each databank.</p> <p>If this <code>-n</code> flag is unspecified, SRS Prisma Quality Report, by default, tests at least 20 entries from each databank on your SRS installation. You can decide how extensive you want the error checking activities to cover your data and change this number by using the <code>-n</code> flag.</p>

Table 7.1 Commandline option flags

Flag	Parameter Name	Type	Default Value	Description
-suites	Specification of Test Suites	String	"	To run tests specified in the test suite. For example, <code>srsinspect -suites 'xml virtual'</code> will tell SRS Prisma Quality Report to run tests in test suites specifically designed for xml and virtual databanks. If this flag is unspecified, all test suites will be run.
-excludeSuites	Specification of Test Suites	String	"	To run tests specified in the test suite. For example, <code>srsinspect -excludeSuites 'xml virtual'</code> will tell SRS Prisma Quality Report to run all test suites except those specifically designed for XML and virtual databanks.
-allTools	Flag to check run tests on all tools	Boolean	False	To run tool-specific tests on all the tools integrated.
-v	Verbosity	Boolean	False	Output messages to trace the progression of SRS Prisma Quality Report in verbose mode.
-d	Diagnose	Boolean	False	Output messages to trace the progression of SRS Prisma Quality Report in diagnostic mode.
-timeOut	Timeout for token test	Int	30	Maximum timeout for token test measured in seconds.

7.4 Error Checking Activities for SRS Relational

Most of the test activities in SRS Prisma Quality Report were written for error checking on flat-file databanks. However, a set of RDB-specific test activities are available for testing SRS installation with Relational Module integrated. Errors

specific to the RDB configuration appear separately in its own class that is color-coded in the graphical HTML output.

7.5 Error Checking Activities for Integrated Tools in SRS

A suite of tools test activities is available in SRS Prisma Quality Report to inspect the integration of tools / applications in SRS. User can use this functionality to check for incorrect configuration after adding common bioinformatics tools like BLAST, FASTA or the EMBOSS package to the SRS server.

7.5.1 Launching the Tools Test

As Tools Test mechanism incorporated in SRS Prisma Quality Report is an activities-intensive process, tools configuration/integration checks are not executed by default unless the

`-allTools` flag is specified when SRS Prisma Quality Report is launched from command line.

e.g. `srsinspect -allTools`

This will launch SRS Prisma Quality Report to check on all the tools integrated on SRS.

User can make use of the `-l` flag to run tool tests only on specified tools selectively. This expedites the process as tool tests are limited to the tools selected. Otherwise, it can result in a long run of SRS Prisma Quality Report if the whole EMBOSS package has been integrated.

e.g. `srsinspect -l 'blastp blastn fasta tfasta hmmsearch' -suites 'tools'`

The above command only executes tool tests on the tools specified in the list of tools given after the `-l` flag; even if there are other tools integrated on this SRS installation.

A Tools Test Results page displays the breakdown summary of errors found in tools; and each tool has an individual Error Details page to describe the nature of the problem and provides suggested actions to correct the problem.

7.6 Error Types Used in SRS Prisma Quality Report

The errors detected by SRS Prisma Quality Report are generally divided into the following main categories:

- Parser errors
- Index errors
- Link errors
- Configuration errors
- Miscellaneous errors
- RDB-specific errors (if SRS Relational is installed)

Each main error category is color-coded for easy viewing.

However, certain error types are sub-divided into categories that better describe the nature of the error. For example, for parser errors, the errors are further divided into the following:

- `misparse` - mismatches in the regular expression
- `empty` - consistently empty token found
- `timeout` - call to parser production time-out
- `duplicate` - duplication of token table

7.7 Quality Report HTML Pages

LION Help Center ?

SRS Prisma

SRS Prisma reports for installation `"/srsuser/support/srs71"`
 Latest Prisma report generated on: Tue Jan 18 11:05:17 GMT 2005

Report Options

SRS Prisma generates 2 different types of reports: Update Reports and Quality Reports

Use the radio buttons to choose the report types you want to view

☒ View update report
☐ View quality report

[View Latest Report](#)

Recent Reports

November 2004

S	M	Tu	W	Th	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

January 2005

S	M	Tu	W	Th	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

December 2004

S	M	Tu	W	Th	F	S
						1
	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Figure 7.1 Calendar Page

Once SRS Prisma has completed its run, users can access the SRS Prima Quality Report HTML Pages via the Calendar Page.

The Quality Report consists of four types of HTML pages:

- main page
- tools test results summary page
- error status change breakdown page
- errors report page for each individual databank or tool

7.7.1 Main Page

The main page gives you the current status of each databank after the latest round of error-checking activities on the day SRS Prisma Quality Report is run. The page is divided into two sections:

- Summary Report
- Databank Group Reports

7.7.2 Summary Report

The first part of the page shows a summary of errors encountered that day on the server.

The summary report shows the following:

- **Today's Error Breakdown** A percentage breakdown in terms of severity score for each category of errors.
- **Databank Independent Errors** A bar chart showing the number of databank independent errors. These are errors that affect the whole system and are not specific to any databanks (those deriving from configuration and miscellaneous categories of errors). Clicking on the color-coded bar will display the Error Details page, which shows databank independent errors in more detail.
- **Overall Error Status Change** An overall error status change on the SRS installation with information on the numbers of databanks with increased/decreased/unchanged error severity respectively. You can click on the highlighted keyword to see the list of database names in each category of changed error severity score (see Figure 7.3).
- **Worst Configured Databanks/Tools** A list of five databanks or tools with the highest error severity score. SRS Administrators are advised to fix the errors in these databanks or tools first.
- **Links to Tools Test Results** If you have run Quality Report with the `-allTools` option, a clickable button to navigate to **Tool Test Results** summary page appears below the Overall Error Status Change section.

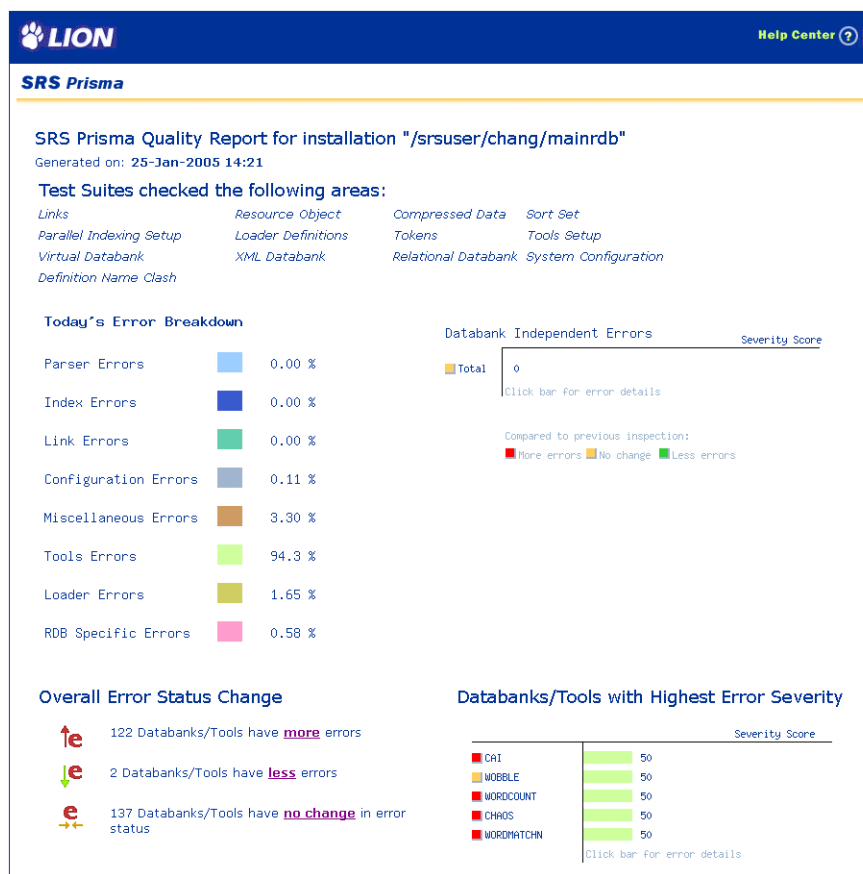


Figure 7.2 Summary Report

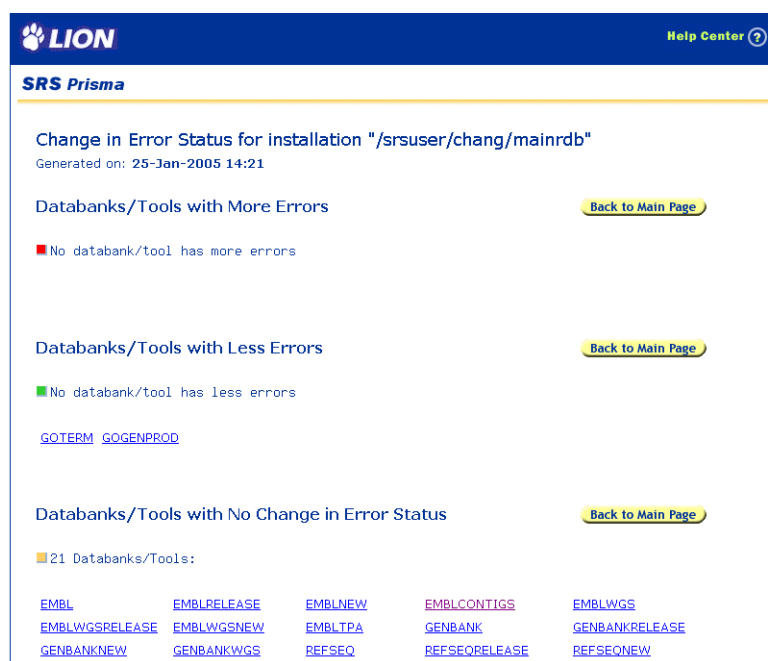


Figure 7.3 Breakdown of Database Names according to their Error Status Change

7.7.3 Databank Group Reports

Following the summary report are reports of the error breakdown of the databanks within each databank group on the SRS installation.

Errors arising from each databank are represented in a color-coded bar chart. Each type of error is given a severity score, the total severity of each databank is represented by the length of the color-coded error bar. The number at the right edge of the color-coded error bar denotes the total severity score. Each category of errors within a databank is represented by a different color, with the length of that color-coded error bar representing the severity score of the corresponding category of errors.

Under each databank group heading, there are two separate charts:

- **Left-Hand Chart** A breakdown of errors and associated error severity score of each databank within the databank group.
- **Right-Hand Chart** A breakdown of errors and associated severity score of a databank within the databank group within the past one month.

7.7.3.1 Left-Hand Chart

The colors of the square box on the left of a databank name reflect the change in error severity for each individual databank between the current and previous runs of SRS Prisma Quality Report. A red square denotes an increase in error severity; a green square denotes a decrease in error severity; and, an amber square denotes no change in error severity of that database.

There are two ways to display the **Error Details** page for a databank:

- Click on the databank name to go the top of the **Error Details** page of that databank, which gives a summary of the status of an individual databank.
- Click on the color-coded bar for a databank will display the specific description of the error type represented by the color within the **Error Details** page, which shows errors identified with the databank in more detail.

Clicking on the color-coded bar for a databank will display the **Error Details** page, which shows errors associated with the databank in more detail.

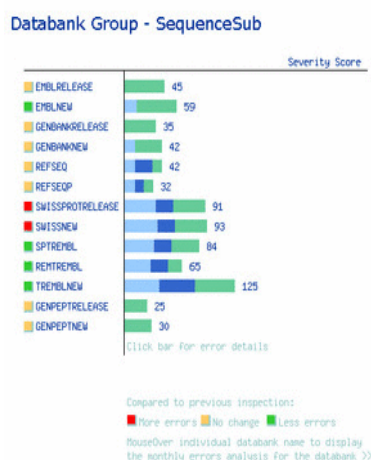


Figure 7.4 Databank Group Report: Left-Hand Chart

7.7.4 Right-Hand Chart

The chart of the right is the breakdown of errors and associated severity score of a databank within the databank group within the past one month.

When you mouse-over (point to) a databank name on the left-hand chart, the chart shows the monthly severity score archive for that databank.

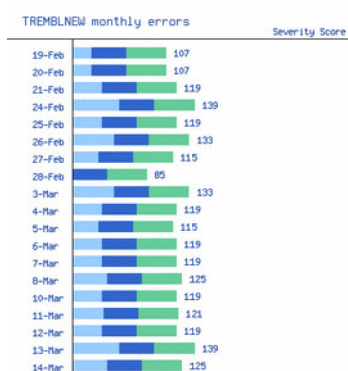


Figure 7.5 Databank Group Report: Right-Hand Chart

7.8 Tools Test Results Page

This is a summary view of the tools test results on the SRS server. The color-coded error severity-bar of different tools is displayed according to the tools groups:

- Alignment Tools
- Display Tools
- Edit Tools
- Nucleic Tools
- Protein Tools
- Phylogeny Tools
- Similarity Search Tools

Users can see the specific set-up errors of a tool by directly clicking on the relevant part of the color-coded error severity-bar of a tool.

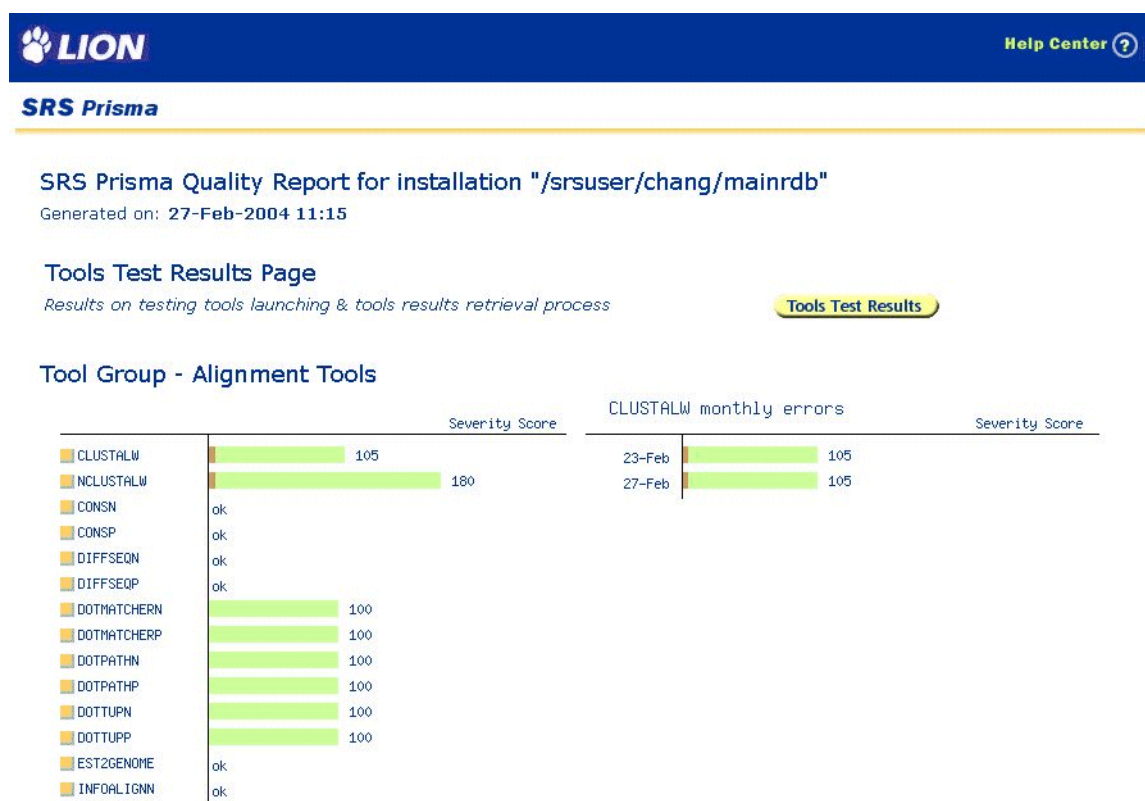


Figure 7.6 Tools Test Results Page

7.9 Error Details Page

There is an error details page for each databank or tool, which can be displayed by clicking on the databank/tool name or the color-coded bar for a databank/tool in a databank/tool group report.

The error details page is divided into several sections:

- Error Summary Report

- Quick References
- Full Error Details Classified by Type Reports
- Monthly Errors Breakdown
- Data-Field Index Status

7.9.1 Error Summary Report

This report shows the combined severity score of each type of error is displayed at the top of the page. In addition, it also shows the comparison between the total severity score of errors arising from the databank/tool after the current round of SRS Prisma Quality Report and that obtained from the previous round.

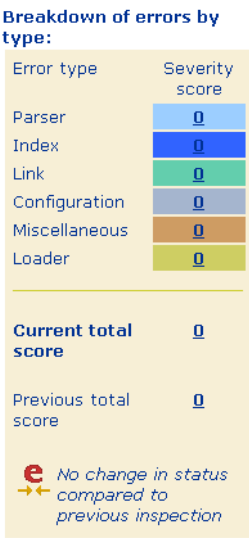


Figure 7.7 Error Summary Report

7.9.2 Quick References Toolbox

This is a toolbox with links to display the various files, which help define the integration of this databank on your SRS installation. If the databank is a relational database, a link to an annotated RDB schema will be given instead.

Non redundant sequence database which combines identical sequences and sub-fragments from the same organism into a single UniRef entry

 [General Databank Information](#)

Structure	.i file	library format and fields definition
Syntax	.is file	parser and token table
Resource	.it file	resource and extra information needed for PRISMA
Settings	srs.gen.i file	field definitions and key internal configuration
\$LibLoc Definitions	srs.db.i file	\$LibLoc definitions of databanks

[View Data Field Index Status for UNIREF100](#)

Figure 7.8 Quick References

7.9.3 Full Error Details Classified by Type Reports

These reports provide full error details specific to each type of error in each databank.



Depending on the nature of the problem, each error is assigned a severity score, and relevant information is also provided to best describe the error.

Most error messages are explicit and self-explanatory on the origin of the problem; furthermore, suggested actions on how to remedy the error are also displayed.



Each error is given a severity score, which gives the SRS Administrator a guide to prioritize error fixing activities during the day-to-day maintenance of SRS server.

Full error details classified by type:

Parser errors

Production	Error type	Error message(s)	Suggested action(s)	Severity
gcgid	empty	Consistently empty token found.	 Check the data source/format, cross reference to the parser syntax in *.is file for this databank. Consider removing this production if data source no longer contains such data.	2
gcgid	timeout	Call to production timed-out.	 Check the data source/format, cross reference to the parser syntax in *.is file for this databank. Consider removing this production if data source no longer contains such data.	10

Index errors

Error message(s)	Suggested action(s)	Severity
The index for the field Title is empty.	 Rebuilding the index for this field is recommended.	10
The Title field has not been indexed or the index has been damaged.	 Reindexing of this field is recommended.	10

Link errors

From databank	To databank	Error message(s)	Suggested action(s)	Severity
REFSEQ	TAXONOMY	Link index from REFSEQ Organism to TAXONOMY Species is empty.	 Rebuilding links between them is recommended.	5
REFSEQ	REFSEQP	Link index from REFSEQ ProteinID to REFSEQP ProteinID is empty.	 Rebuilding links between them is recommended.	5

Figure 7.9 Full Error Details Classified by Type Reports

7.9.4 Monthly Errors Breakdown

Near the bottom of the **Error Details** page for each databank, you can find a bar chart on the monthly breakdown of archived error severity score. By clicking on the error bar specific to a certain date in the past one month, user will be directed to the **Error Details** page for that date selected. This gives you a chance to track the changes in errors specific to each databank in the past one month.

Monthly errors breakdown

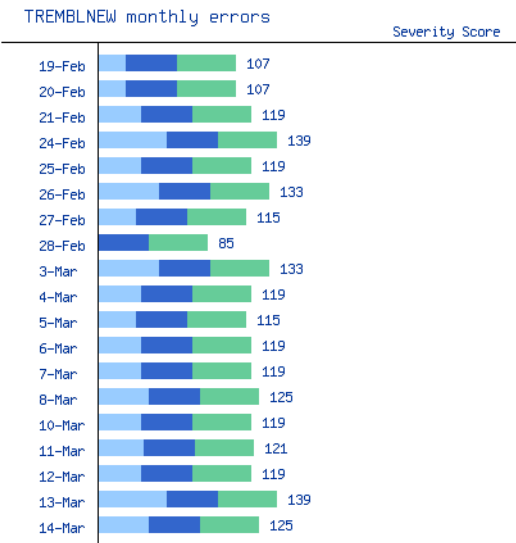


Figure 7.10 Monthly Error Breakdown

7.9.5 Data-Fields Index Status

A table of the status of each indexed data-field is shown at the bottom of the **Error Details** page for each databank.

Data-Field Index Status for ENZYME

Name	Short Name	Type	No.of Words	No.of IDs	Time Created
AllText	all	group	0	0	-
ID	id	id	4208	4208	26-Feb-2004 08:12
AltName	alt	index	4888	9361	26-Feb-2004 08:12
Description	des	index	3961	8902	26-Feb-2004 08:12
CatalyticActivity	cat	index	5462	28671	26-Feb-2004 08:12
Comment	cc	index	0	0	-
Link	link	link	0	0	-

File indexed :

Name	File Size	File Location
enzyme.dat	2221120	/srsdata/flatfiles/data8/enzyme/enzyme.dat

Figure 7.11 Data-Fields Index Status

7.10 Archiving

Upon the completion of Prisma update processes and subsequent error-checking activities encoded by Quality Report, Prisma will automatically archive the Update Report and Quality Report in the directory structure predefined by \$REPORTDIR.

The Prisma Calendar page (Figure 7.1 on page 201) provides a user-friendly interface to navigate to both current and archived Update Report and Quality Report. Depending on the option button selected, you can choose to view the current and archived Update Report main page (Section 5.3.4, Update Report Page, page 153) or Quality Report main page (Section 5.3.16, The Quality Report, page 187).

While you are in an archived Update Report main page (Figure 5.7 on page 155), you can also access the archived Quality Report Error Details page showing the error status for a databank for that day of interest, by clicking on the **View Quality Report** button for that databank.

7.11 Tests Used

Error-checking activities in each test suite are listed below:

Table 7.2 Test Suite LINKS

Test Name	Description
<code>checkLinkSetup</code>	Checks for inappropriately defined links (links that are a mix of a read-link and an index-link).
<code>linkToUndefinedDB</code>	Checks that no link is created such that it points to a non-existing databank.
<code>emptyLinkIndices</code>	Looks for index-link indices that are empty. Also checks that the databanks to which links are defined actually exist.

Table 7.3 Test Suite RESOURCE

Test Name	Description
<code>resource object</code>	Checks that an <code>.it</code> file exists for a databank. If the name of the <code>.it</code> file is not specified in the databank object, it is manually constructed from the databank name and an <code>.it</code> extension.

Table 7.4 Test Suite COMPRESSION

Test Name	Description
<code>compressedData</code>	General checks for configuration to handle compressed data when data compression is applied to a database during indexing

Table 7.5 Test Suite SORTSET

Test Name	Description
<code>checkSortSet</code>	Check that each field that is made sortable has the corresponding <code>sort-set.om</code> in the index directory for flatfile databases.

Table 7.6 Test Suite PARALLEL

Test Name	Description
FileParallelisation	Checks that indexing for databanks with only one datafile are not parallelized using the <code>file</code> method.
ChunkParallelisation	Checks that indexing for databanks with more than one datafile are not parallelized using the <code>chunk</code> method.
ChunkSizeParallelisation	Checks that indexing for databanks with more than one datafile are not parallelized using the <code>chunkSize</code> method.

Table 7.7 Test Suite LOADER

Test Name	Description
LoaderLibTest	Checks that library associated with a loader actually exists.
LoaderGroupTest	Checks that library group associated with a loader actually exists.

Table 7.8 Test Suite TOKENS

Test Name	Description
checkToken	This activity creates a system call to execute <code>\$SRSLION/prisma/tokenTest.i</code> via <code>gtime</code> . The output from <code>gtime</code> and/or <code>tokenTest.i</code> is retrieved and parsed to find out what happened.

Table 7.9 Test Suite TOOLS

Test Name	Description
ToolLaunch	Checks that the tool integrated can be launched properly.

Table 7.10 Test Suite VIRTUAL

Test Name	Description
VirtualMembersCheck	Checks for members of virtual databanks. Make sure that member databanks defined for each individual virtual library do exist.
VirtualQuerySetCheck	Checks that virtual query set of virtual databanks has been built.
VirtualQueryStrCheck	Checks whether the virtual query string of virtual databanks is defined properly within <code>virtualInfo</code> of databank object in the <code>.i</code> file.

Table 7.11 Test Suite XML

Test Name	Description
XmlMetaphorCheck	Extract XML metaphors for XML database specific tests.

Table 7.12 Test Suite RDB

Test Name	Description
checkSchemaName	Check that the schema name given in the <code>schemas</code> attribute of <code>\$RdbLibHub</code> matches that written at the beginning of the <code>schema.i</code> file.
checkRDBTableOwner	Check that the name given in the <code>schemas</code> attribute of <code>\$RdbLibHub</code> (<code>i.file</code>) matches the <code>\$RdbTable</code> 'owner' attribute in the <code>schema.i</code> file.
foreignKeyCheck	Check that the foreign key is pointing to an existing column in the specified parent table correctly.
rdbFieldCheck	Check that <code>srsFields</code> in <code>RDBSchema</code> are fully utilized in <code>.i</code> file.
checkFieldToSchema	Check that <code>srsFields</code> defined in <code>.i</code> file exists in <code>RDBSchema</code> in <code>schema.i</code> file.

Table 7.13 Test Suite NAMECLASH

Test Name	Description
LibGroupNameClash	Check that the name of a library group is not identical to any of the existing library group or library name.
LibGroupShortNameClash	Check that the short name of a library group is not identical to any of the existing library group or library name.

Table 7.14 Test Suite SYSCONFIG

Test Name	Description
checkAdminEmail-Address	Checks that the e-mail address of SRS Administrator is in the format one would expect for e-mail addresses.
checkGunzipLocation	Checks that the location of the gunzip executable has been set correctly.

CHAPTER

8

CONFIGURATION UPDATES WITH SRS PRISMA

8.1 Introduction

SRS is a highly customizable and extensible data integration platform, and as such, relies on extensive meta-data files defining aspects of the data and tools integrated by the installation. By their very nature, the underlying data sources are subject to change by data providers, and hence new meta-data files are frequently required. However, the customizable nature of SRS means that local changes are frequently made to these files, and the process of checking for new meta-data files and importing to is a tedious one.

With the introduction of SRS Prisma 4.1, this process can be automated, with minimal intervention required from the administrator. The configuration update mechanism in SRS Prisma 4.1 is designed to automatically check for updates to Icarus and other files using the LION parser file repository to find and download the latest files, and merge the new files with any local customisations.

8.1.1 The configuration file update mechanism

The configuration file update mechanism is used to maintain files within a specified directory of an SRS installation with reference to a directory on a remote repository. To allow local customisations to be maintained, two local directories are used. The **distribution** directory (e.g. SRSDb) is used to maintain a local reference copy of files distributed as part of SRS. The **working** directory (e.g. SRSSITE) contains working copies of these files, which may be modified as part of a local customisation. The **remote** directory is a remote resource where new copies of configuration files are deposited. The following directories are configured by default to be updated by SRS Prisma 4.1

Table 8.1 Configuration directories updated by SRS Prisma

Config- uration	Distribution directory	Working directory	Remote repository
srsdb	\$SRSDb	\$SRSSITE	http://downloads.lionbio.co.uk/parser/srs81/ icarus/db/
emboss	\$SRSLION/ emboss_dist/	\$SRSLION/ emboss/	http://downloads.lionbio.co.uk/parser/srs81/ icarus/lion/emboss/
jsp	\$SRSROOT/srstags/ srstags/web/	\$SRSROOT/ srstags/ srstags_dist/ web/	http://downloads.lionbio.co.uk/parser/srs81/ srstags/srstags/web/

The configuration file update process is as follows (illustrated in Figure 8.1):

check and download new files

The remote update module of SRS Prisma is used to carry out a comparison of the distribution and remote directories. New files are downloaded into the corresponding sub-directory of the “offline” SRS hierarchy (\$SRSPRISMA/updateConfig/srs_off/) and files that do not need updating are copied into this directory from the online distribution directory.

compare new and existing files

If new files have been found, they are compared to the existing online files. If a modified copy is found in the working directory, a three-way comparison is carried out between the online distributed and working copies and thenew copy to identify if the new and working files can be merged without conflicts. The comparison is carried out by Icarus object-based comparison if the files are already referenced by SRSSITE:srsdb.i, and contain Icarus object definitions. For all other files, the diff3 file comparison tool is used for textual comparison. The comparison process flags each new file as one of the following:

- **replace** - the distributed copy of the file should be replaced.
- **replaceSite** - both the distributed and working copies of the file should be replaced as they are identical.
- **update** - the distributed copy should be replaced, and the working copy should be merged with the new copy

- **conflict** - as for update, but the merge is not possible without manual intervention.

resolve conflicts

If unresolved conflicts have been found, the administrator must manually run a conflict resolution tool to identify whether the new change or existing modification should be used. Note that once the resolution has been decided, SRS Prisma can “remember” the decision for the file or object in question so that persistent local modifications can be automatically selected in preference to changes to the distributed version, without the need for manual intervention. These decisions are read during the comparison step (see above).

create merged offline set

Once any conflicts have been resolved, the merging process systematically applies new changes and existing modifications to new and existing files to create a complete offline working directory, and uses `srsection` to create an offline SRSOM directory that can be used by SRS. In addition, if new files have been flagged as requiring reindexing of SRS libraries (e.g. if a parser changes how a field is indexed), the appropriate indices are “touched” so they will be identified as required reindexing when next checked by SRS Prisma or `srsccheck`.

update libraries

If the update mechanism has been run from within the main SRS Prisma update process (see Figure 8.2.2), and no conflicts have been found, control passes back to the main `runPrisma` process. `runPrisma` now switches to using the newly created offline configuration file set and SRSOM. The checking process continues as normal, and any updating will use the new parsers and configuration files. Note that all other SRS processes continue to use the existing online files, ensuring complete compatibility between data, indices and configuration files. If the updating process completes without errors, the next phase moves the new configuration files online, as described below.

move files online

Once all previous updating processes have completed successfully, the new configuration files are moved online. Before the move phase occurs, the original working and distribution directories are backed up to allow rapid restoration of previous settings. The files are moved online into the distribution and working

directories as appropriate, and srsection is run to compile new object files for use by the rest of SRS.

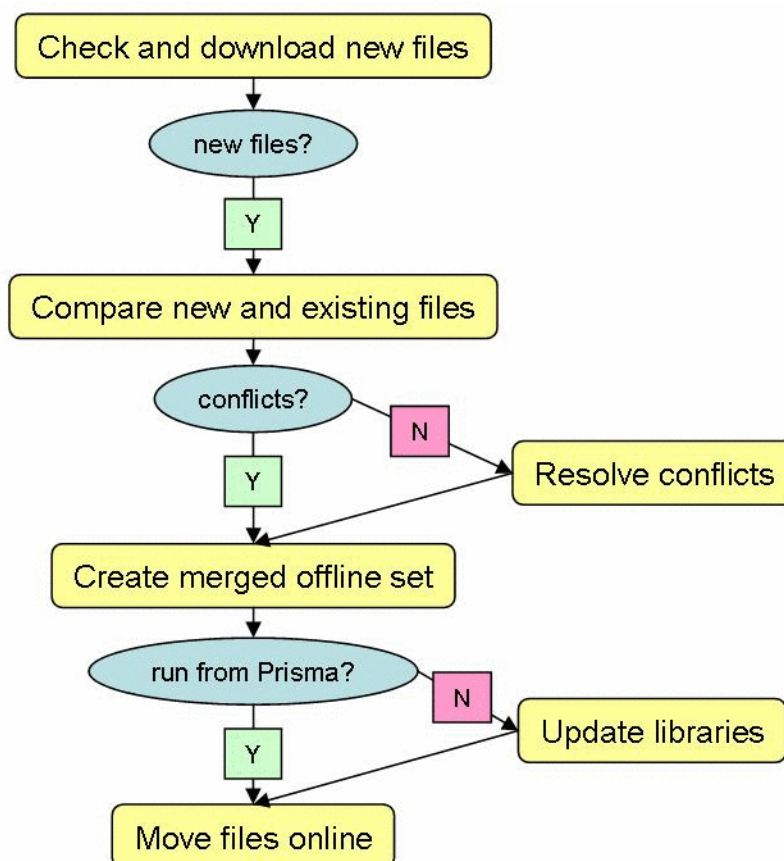


Figure 8.1 The configuration update process

8.2 Using the update mechanism

The update mechanism can be used in one of two different ways, according to the requirements of the local installation. The first allows the update mechanism to be run as a standalone mechanism, and the second as an integrated part of the Prisma update mechanism. It is recommended that the first method be used for heavily customised installations where the possibility of conflicts.

Important: Whatever mechanism is used, the configuration files being updated must not be changed by an external mechanism during the process or the results will be unpredictable. The update mechanism will not start if the Visual Administration tool server is running.

8.2.1 Running as standalone

The configuration update mechanism can be run as an independent process with the following command

```
% updateConfigFiles.sh
```

Note that this process will not run if another instance of `updateConfigFiles.sh` is running, as evinced by the lockfile `$SRSPRISMA/PRISMA_CONFIG_RUNNING`.

The following shows some sample output of a successful run:

```
[2005/01/11 11:49] Updating configuration files
[2005/01/11 11:49] Starting check and download of new configuration files
[2005/01/11 11:49] 3 new files downloaded
[2005/01/11 11:49] Starting comparisons of new and existing files
[2005/01/11 11:49] Comparisons of new and existing files complete
[2005/01/11 11:49] Checking for conflicts with current configuration
[2005/01/11 11:49] Merging new and existing configuration files
[2005/01/11 11:50] Merge of configuration files complete
[2005/01/11 11:50] Moving updated configuration files online
[2005/01/11 11:50] Moving files online
[2005/01/11 11:50] Compiling online files
----- Welcome to SRS 8.1 -----
[2005/01/11 11:50] Generating reports
```

If conflicts are found, the process will not continue, and the output will inform the user that manual intervention is required:

```
[2005/01/11 11:57] Updating configuration files
[2005/01/11 11:57] Starting check and download of new configuration files
[2005/01/11 11:57] 3 new files downloaded
[2005/01/11 11:57] Starting comparisons of new and existing files
```



```
[2005/01/11 11:57] Comparisons of new and existing files complete
[2005/01/11 11:57] Checking for conflicts with current configuration
[2005/01/11 11:57] ERROR: Conflicts found during update process.
Conflicts found during update process
[2005/01/11 11:57] ERROR: Please resolve these by running
${SRSETC}/updateConfigFiles.sh -resolve
[2005/01/11 11:57] Generating reports
```

The following command should be run to manually resolve the conflicts:

```
% updateConfigFiles.sh -resolve
```

Conflict resolution is discussed in more detail in Section 8.2.3, Conflict Resolution, page 225.

To continue the update process, run:

```
% updateConfigFiles.sh -continue
```

(-continue can be combined with -resolve if required)

The process as described will update and install new configuration files. If any of these files require reindexing of SRS libraries, the next run of SRS Prisma or srscheck will identify fields that need reindexing:

However, in the intervening time, the configuration files may not match the current data and indices.

As an alternative strategy, the new files can be left offline and used for any required reindexing, using the following commands:

```
% updateConfigFiles.sh -noMove
```

This creates a complete offline SRS configuration file set. To make any subsequent SRS processes launched from the current shell use these files, one of the following command can be used:

```
% . $SRSLION/prisma/configUpdate/prep_srs_offline.sh
% source $SRSLION/prisma/configUpdate/prep_srs_offline
```

Indexing can then take place e.g.:

```
% runPrisma -local
```

Once indexing has been carried out successfully, the following command can be used to install the new files:

```
% updateConfigFiles.sh -doMove
```

Further to the flags discussed above, `updateConfigFiles.sh` accepts the following command line flags:

Table 8.2 Command line flags for `updateConfigFiles.sh`

Flag	Type	Default Value	Description
-configName	string	""	Restricts checking to the named configuration directory (by default one of <code>srsdb</code> , <code>emboss</code> or <code>jsp</code>)
-noMove	boolean	FALSE	Update and merge the configuration files but do not move online
-doMove	string	FALSE	Move completed offline configuration files online
-resolve	boolean	FALSE	Run the manual conflict resolution process
-continue	boolean	FALSE	Continue the update process after manually resolving conflicts
-restore	boolean	FALSE	Restore the original configuration files from before the previous run
-v	boolean	FALSE	Print verbose output
-report	boolean	FALSE	Regenerate reports only

8.2.2 Running as part of Prisma

The update process can also be run as part of the standard Prisma update process, either launched directly as `runPrisma.sh` or via `launchPrisma.sh`. This will happen automatically if configured (see Section 8.4.1, “Basic configuration”, p. 235), or if the `-updateConfigFiles` flag to `runPrisma` is used.

The process runs the following steps:

- download and create complete offline set

- switch to using offline configuration set
- carry out normal SRS Prisma update process (forcing reindexing where required)
- move new configuration files online
- move new data and indices online

If unresolved conflicts are found, runPrisma will report their presence and exit. At this point, the following command should be run to resolve these conflicts:

```
% updateConfigFiles.sh -resolve -nomove
```

The next runPrisma process to run will then use the offline configuration files as before.

If the configuration update process fails, the normal Prisma update process will not take place. Conversely, if the normal Prisma update process fails for any reason, the new configuration files will not be moved online, and the entire Prisma installation process will be blocked, even for libraries that succeeded. This minimises the risk of incompatible or corrupt files being installed. However, data and indices can be moved online using:

```
% movePrisma.sh -run <runDate>
```

The updated configuration files can be moved online manually using:

```
% updateConfigFiles.sh -doMove
```

If any indexing is required, it will be carried out by the next Prisma run.



Important: When automatic configuration updates are run as part of the main Prisma process, library update schedules (see Section 3.12.1, Scheduling for Automatic Updates, page 113) **MUST** include indexing on all days that Prisma runs. This is to support the forced reindexing of libraries that some configuration updates may require.

8.2.3 Conflict Resolution

When conflicts are encountered that require manual intervention, the following command can be used to resolve them:

```
% updateConfigFiles.sh -resolve
```

This command is interactive, and asks for the resolution for each conflict found. The following example shows a typical resolution session for a conflict found in the `partSizeKb` attribute of the `EMBLNEW_DB` object in `SRSDb:embl.i`:

```
[2005/01/11 12:42] Resolving conflicts with existing files
[Info] File is object based
[Info] Unresolved conflict(s) found in emblnew.i
[Info] Conflict found:
To keep existing:
Object:  EMBLNEW_DB.partSizeKb
Action:  define
Value:   300000
To upgrade to new:
Object:  EMBLNEW_DB.partSizeKb
Action:  define
Value:   150000

1 : Resolve conflict using new value
2 : Resolve conflict using existing value
Enter choice : [ 1 ]
[Info] SRS can remember this decision, and use it to automatically resolve
conflicts involving the whole object, or specific attributes of the object.
Would you like SRS to remember this choice for the object EMBLNEW_DB in future? : [
y ] n
Would you like SRS to remember this choice for the attribute EMBLNEW_DB.partSizeKb
in future? : [ y ] y
```

The response given here instructs SRS Prisma to replace the existing value (300000) with the new value (150000).

Next, SRS Prisma can remember whether a decision was made to keep or replace values for this object and attribute:

The responses given here indicate that conflicts should still be resolved manually for the `EMBLNEW_DB` object in general, but any conflicts involving `EMBLNEW_DB.partSizeKb` should be automatically resolved using the new distributed value. These decisions can be edited at a later date if desired (see Section 8.4.6, “Editing memorized decisions”, p. 241).

The following output shows resolution of a conflict involving the embl.is parser file, which has been compared on a textual basis:

```
[Conflict] Found in embl.is
====
/opt/software/srs/icarus/site/embl.is:402c
    /[A-Z0-9]+/ {$Rep:$Hlink:[localPatentR p:{$Ct $Ct}]}
/opt/software/srs/icarus/db/embl.is:402c
    /[A-Z0-9]+/ {$Rep:$Hlink:[patentR p:{$Ct $Ct}]}
/opt/software/srs/prisma/updateConfig/srs_off/icarus/db/embl.is:402c
    /[A-Z0-9]+/ {$Rep:$Hlink:[newPatentR p:{$Ct $Ct}]}

    1 : Resolve conflict using new file
    2 : Resolve conflict using existing file
    3 : Install new file without merging
    4 : Keep existing file without merging
Enter choice : [ 1 ] 2
[Info] SRS can remember this decision, and use it to automatically resolve
conflicts involving this file.
Would you like SRS to remember this choice for the file embl.is in future? : [ y ] n
```

The responses given here instruct SRS to use the local value when creating the new file, but not to automatically resolve conflicts involving embl.is in future, as there may be other conflicts that need attention and the decision applies to the entire file.

8.2.4 Restoring previous configuration files

Following an update process, it may be found that the changes are not desirable, or cause problems. To rapidly reverse the changes made in the last run, the following command can be used:

```
% updateConfigFiles.sh -restore
```

The -configName flag may also be used to restrict the reversion to one particular directory although this should be used with care as frequently new JSP view files require updated configuration files from SRSDb.

Note: It is also possible to manually revert changes - the original directories are named `<configName>_bak` e.g. `srs/icarus/db_bak`. However, take care to ensure that the original file dates are preserved (e.g. using `cp -p`) or the update process may not work reliably the next time it is run.

Important: The restore functionality should be used with care if indexing has been carried out using new parsers and configuration files, since there may be mismatches between the indices and configuration files.

8.2.5 Importing Icarus files manually

In addition to the automated mechanism described here, SRS Prisma also supplies a command line tool for manually importing an individual Icarus file into SRSDb, allowing merging, conflict resolution etc.

The following command imports a new Icarus file:

```
% ${SRSLION}/prisma/configUpdate/importConfigFile.i -fileName /tmp/embl.i
[Update] Action: update
[Info] Applying updates for embl.i
[Update] Applying changes to objects in "site" copy
[Update] New version of embl.i successfully installed
```

If conflicts are found, the command prompts for their resolution as described in Section 8.2.3, “Conflict Resolution”, p. 225.

Important: This tool currently only works for files that are found in SRSDb/SRSSITE, and not in other directories.

8.3 Update Reports

In addition to the text output from the command line tool, SRS Prisma produces HTML reports when it updates configuration files. These can be accessed from the main SRS Prisma Calendar page by selecting the “Show configuration report” option and clicking on “Show Latest Report” or clicking on an archived number, as shown in Figure 8.2:

Figure 8.2 The SRS Prisma Calendar page

The screenshot displays the SRS Prisma web interface. At the top is a blue header with the LION logo and a 'Help Center' link. Below the header, the page title 'SRS Prisma' is shown. The main content area has a subtitle 'SRS Prisma reports for installation "/opt/software/srs"' and a message 'Latest Prisma report generated on: Thu Jan 6 16:53:31 GMT 2005'. On the left, a 'Report Options' box explains that two types of reports are generated and provides radio buttons to select between 'View update report', 'View configuration update report' (which is selected), and 'View quality report'. A 'View Latest Report' button is at the bottom of this box. To the right, 'Recent Reports' are shown as three calendar grids for January 2005, December 2004, and November 2004. Each calendar has a header with the month and year, and a table of days of the week and dates. In the January 2005 calendar, the date 1 is highlighted in blue. In the December 2004 calendar, the date 15 is highlighted in blue. In the November 2004 calendar, the date 1 is highlighted in blue. At the bottom of the page, a small copyright notice reads: 'SRS Release 8.1 Copyright © 1997-2005 LION Bioscience AG. All Rights Reserved. Terms of Use'.

SRS Prisma

SRS Prisma reports for installation "/opt/software/srs"
Latest Prisma report generated on: Thu Jan 6 16:53:31 GMT 2005

Report Options

SRS Prisma generates 2 different types of reports: Update Reports and Quality Reports

Use the radio buttons to choose the report types you want to view

☐ View update report
☒ View configuration update report
☐ View quality report

View Latest Report

Recent Reports

January 2005

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

December 2004

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

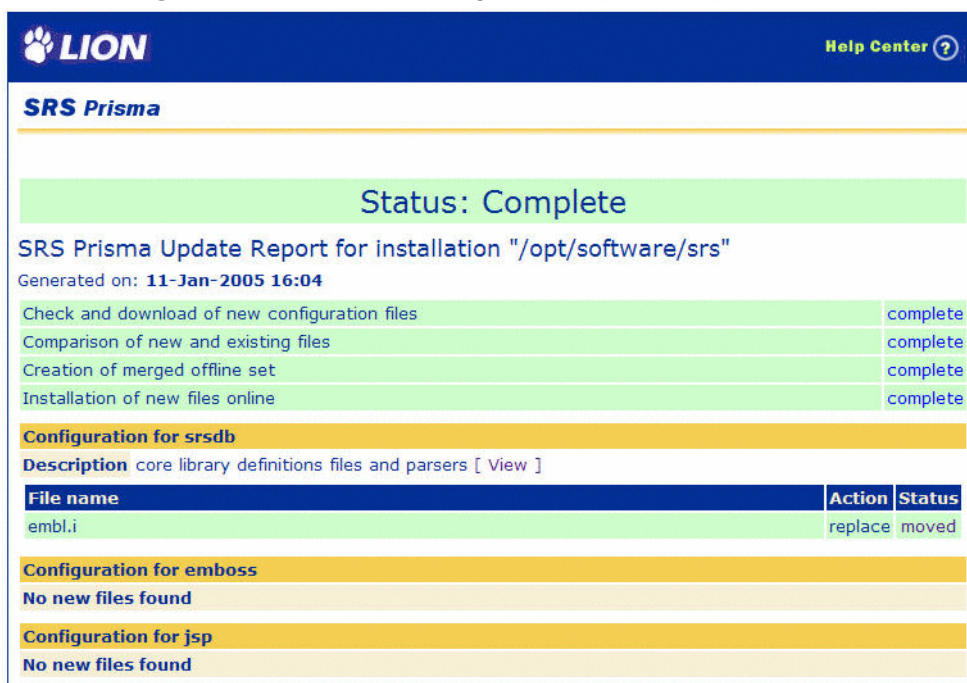
November 2004

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

SRS Release 8.1 Copyright © 1997-2005 LION Bioscience AG. All Rights Reserved. Terms of Use

Choosing to view a Configuration Report takes you to the main configuration report page. An example is shown in Figure 8.3:

Figure 8.3 The main configuration update report



The report is divided into two main sections. The first section shows the overall progress, with color coding indicating the status of each stage. In Figure 8.3, all

stages have completed successfully. In Figure 8.4, the comparison stage shows a conflict has been found:

Figure 8.4 The main configuration report for a failed update

LION [Help Center ?](#)

SRS Prisma

Status: Failure

SRS Prisma Update Report for installation "/srsuser/staines/srs"

Generated on: **13-Jan-2005 12:59**

Check and download of new configuration files	complete
Comparison of new and existing files	complete
Creation of merged offline set	failed

Configuration for srsdb

Description core library definitions files and parsers [View]

File name	Action	Status
emblnew.i	conflict	conflict (resolved)
embl.i	update	new
embl.is	conflict	conflict (unresolved)

Configuration for emboss

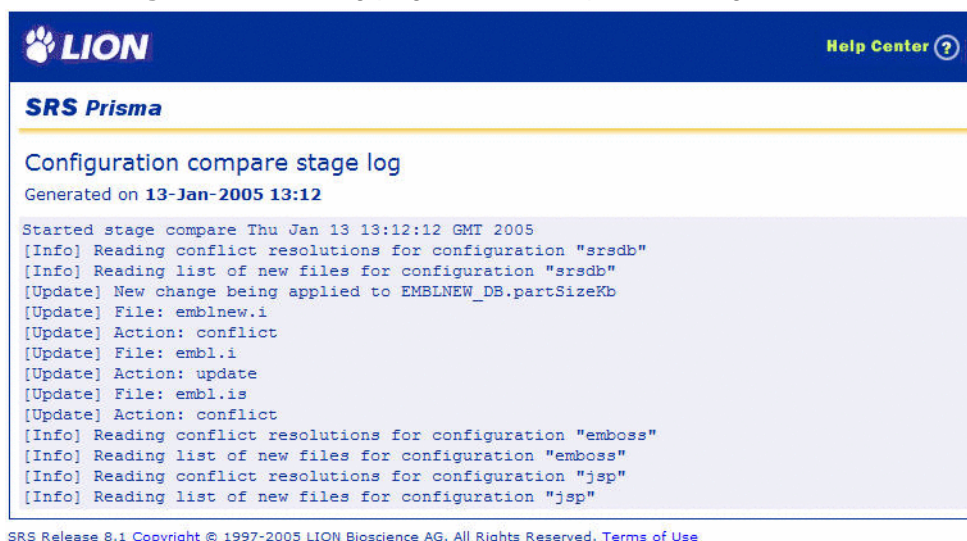
No new files found

Configuration for jsp

No new files found

The logs for each stage can be examined by clicking the hyperlink in the stage summary. Figure 8.5 shows the log page for the comparison stage:

Figure 8.5 The log page of the comparison stage



The next section of the main report page is divided into sub-sections for each configuration update directory. In each section, the status of new files are shown. This status can be one of:

new

the file has been downloaded but not yet compared and merged

updated

the file has been successfully merged with the online copy/copies

moved

the file has been moved online

conflicts (resolved)

the file contains a conflict which has since been resolved either manually or automatically

conflicts (unresolved)

the file contains a conflict which requires manual intervention

In Figure 8.4, one file has been successfully updated, one has been automatically resolved but another file requires manual conflict resolution:

In each case, the status can be clicked to view a detailed report page, as illustrated in Figure 8.6:

Figure 8.6 File report for a file with unresolved conflicts

The screenshot displays the SRS Prisma web interface. At the top, there is a blue header with the LION logo and a 'Help Center' link. Below the header, the title 'SRS Prisma' is shown. The main content area is divided into several sections:

- General Information:** A table with two rows: 'File name' (embl.is) and 'Action' (conflict - Unresolved).
- Distributed copy of file:** A table with two rows: 'File status' (new) and 'Online location' (/srsuser/staines/srs/icarus/db).
- Working copy of file:** A table with two rows: 'File status' (conflicts) and 'Online location' (/srsuser/staines/srs/icarus/site).
- Update Details:** A section containing three patches: 'Difference:', 'Upgrade patch:', and 'Modification patch:'. Each patch shows a list of files and their corresponding links.

The 'Update Details' section contains the following text:

```

Difference:
====
/srsuser/staines/srs/icarus/site/embl.is:402c
/[A-20-9]/ ($Rep:$Hlink:[localPatentDbR p:{$Ct $Ct}])
/srsuser/staines/srs/icarus/db/embl.is:402c
/[A-20-9]/ ($Rep:$Hlink:[patentR p:{$Ct $Ct}])
/srsuser/staines/srs/prisma/updateConfig/srs_off/icarus/db/embl.is:402c
/[A-20-9]/ ($Rep:$Hlink:[newPatentR p:{$Ct $Ct}])

Upgrade patch:
402c
/[A-20-9]/ ($Rep:$Hlink:[newPatentR p:{$Ct $Ct}])
.

Modification patch:
402c
/[A-20-9]/ ($Rep:$Hlink:[localPatentDbR p:{$Ct $Ct}])
.

```

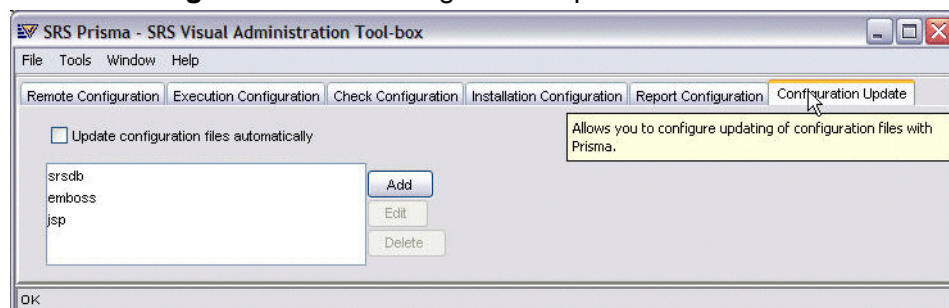
SRS Release 8.1 Copyright © 1997-2005 LION Bioscience AG. All Rights Reserved. Terms of Use

8.4 Configuration of the update mechanism

As with all aspects of SRS Prisma, the configuration update mechanism is highly configurable, allowing additional configuration update directories to be added, and for fine tuning of the set-up for the default directories.

To edit the configuration for the update mechanism, start the Visual Administration tool, open the SRS Prisma tool. All configuration can then be accessed through the **Configuration Update** tab, as shown in Figure 8.8:

Figure 8.7 The Configuration Update tab of VisAd



It is recommended that SRS VisAd is used to edit configurations, but experienced Icarus users can also directly edit the `$$SRSETC/conf/prisma` file directly. The relevant configuration is an instance of the `PrismaConfigSettings` class, set in the `configSettings` attribute of the main `PrismaSettings` object

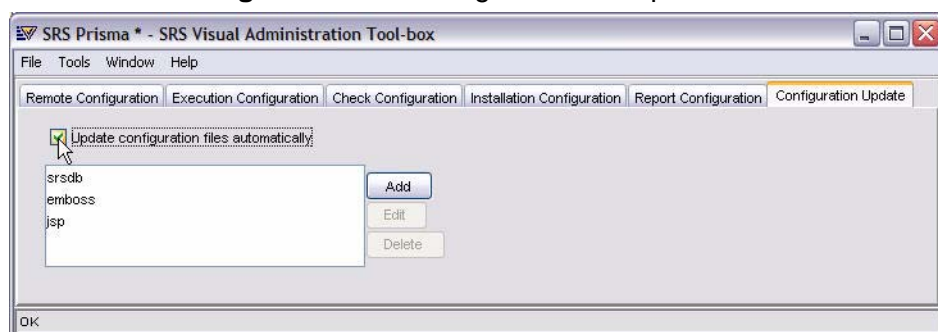
```
$prismaConfigObj=$PrismaSettings:[ ..
configSettings:$PrismaConfigSettings:[
  updateConfigFiles:y
  remoteConfigs:{
    $RemoteUpdateConfig:[name:'srsdb'
      description:'core library definitions files and parsers' alwaysCopy:n
      remoteFileLocations:$RemoteFileLocation:[name:'lion_srsdb' isDefault:y
      remoteHosts:$RemoteHost:[hostName:"downloads.lionbio.co.uk"
      downloadDir:"/srs81/icarus/db/"
    ]
    remoteFiles:$RemoteFilePattern:[searchPatternRemote:'.i[ltsc]?\'$'
      searchPatternLocal:'.i[ltsc]?\'$'
    ]
  ]
  conflictResolutions:$ConflictResolution:[name:'EMBLNEW_DB.partSizeKb'
    action:replace
  ]
]
..
}
```

In each case described below, the corresponding Icarus classes and attributes are described.

8.4.1 Basic configuration

By default, the configuration update mechanism does not run automatically during an SRS Prisma update. This can be changed by checking the **Update configuration files** automatically box in the **Configuration Update** tab.

Figure 8.8 Activating automatic updates



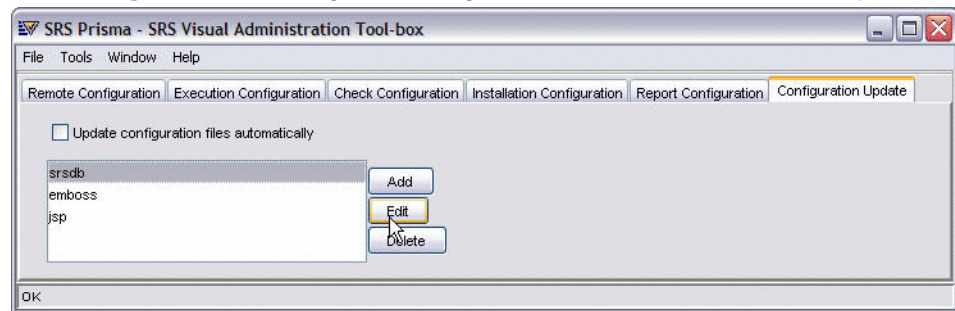
The same behavior may be controlled with the `PrismaConfigSettings.updateConfigFiles` attribute.

8.4.2 Editing remote configurations

Each directory that is updatable by SRS Prisma has its own configuration. An existing configuration can be deleted or edited by selecting its name in the list, as shown in

Figure 8.9. These can also be edited by adding, deleting or editing `RemoteConfig` objects in the `PrismaSettings.remoteConfig.remoteConfigs` attribute.

Figure 8.9 Editing the configuration for the 'srsdb' directory



Creating a new remote configuration or editing an existing one opens the Remote Configuration editor, which is shown in Figure 8.10. The following properties can be altered (`RemoteConfig` attributes shown in parentheses):

configuration name (*name*)

a descriptive, single word name that can be used to reference the configuration

Description (*description*)

a brief description of the directory and its contents

Carry out check as default (*check*)

whether the directory should always be checked. If not set, the `-configName` flag to `runPrisma.sh` or `updateConfigurationFiles.sh` must explicitly name this configuration.

Online directories (*onlineDist*, *onlineSite*)

Locations of the online distribution and working directories to mirror.

Offline directories (*offlineDist*, *offlineSite*)

Locations of the temporary offline directories to store. These should be set to the corresponding locations beneath `$SRSPRISMA/configUpdate/srs_off/` e.g. `$SRSPRISMA/configUpdate/srs_off/icarus/db/`. This allows any Icarus files to be read and merged as objects by the merge mechanism.

Mirror sub-directories (*recursive*)

If this option is set, all sub-directories beneath the specified directory will be mirrored.

Distribution copy is not read by SRS (*alwaysCopy*)

This option should normally be set, as it creates a parallel distribution directory to be used as a reference against which the current working directory can be compared. This means that the distribution directory is always a mirror of the remote site (but is not read by SRS), but the working directory also contains all files, containing any local changes read by SRS. This is not used for the SRSSITE/SRSDB set-up, where SRSSITE need only contain a subset of files from SRSDB.

Remote mirror locations (*remoteLocations*)

This contains a list of remote locations (see Section 8.4.3, “Editing remote mirror locations”, p. 239) which describe the remote hosts and files used to maintain a directory.

Files to be merged as text (*noObjMerge*)

This is a list of regular expressions matching files that should always be merged as text, rather than as objects. This is needed for Icarus files that contain programmatic constructions rather than plain object definitions.

Files which should not be merged (*noMerge*)

This is a list of regular expressions matching files that should never be merged, but should always be replaced wholesale.

Automatic conflict resolution (*conflictResolutions*)

This is a list of conflict resolution rules (see Section 8.13, “Editing a remote file pattern”, p. 241) which are used to automatically resolve conflicts.

Figure 8.10 Editing the configuration for the ‘srsdb’ directory

The screenshot shows a dialog box titled "Edit details of remote configuration update" with a close button (X) in the top right corner. The dialog is organized into several sections:

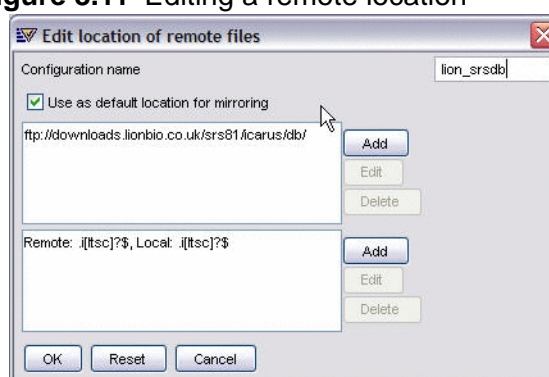
- Configuration name:** A text field containing "srsdb".
- Description:** A text field containing "core library definitions: files and parsers".
- Carry out check as default:** A checked checkbox.
- Directories to mirror:**
 - Online directories:**
 - Distribution:** A text field with "/opt/software/srs/icarus/db/" and a "Browse" button.
 - Working:** A text field with "/opt/software/srs/icarus/site/" and a "Browse" button.
 - Offline directories:**
 - Distribution:** A text field with ":ware/srs/prisma/updateConfig/srs_off/icarus/db/" and a "Browse" button.
 - Working:** A text field with "ware/srs/prisma/updateConfig/srs_off/icarus/site/" and a "Browse" button.
 - Mirror sub-directories:** An unchecked checkbox.
 - Distribution copy is not read by SRS:** An unchecked checkbox.
- Remote mirror locations:**
 - A list box containing "lion_srsdb".
 - Buttons: "Add", "Edit", and "Delete".
- Explicit merging rules:**
 - Files to be merged as text:** An empty text area with "Add", "Edit", and "Delete" buttons.
 - Files which should not be merged:** An empty text area with "Add", "Edit", and "Delete" buttons.
- Automatic conflict resolution:**
 - A text field containing "Always replace EMBLNEW_DB.partSizeKb".
 - Buttons: "Add", "Edit", and "Delete".

At the bottom of the dialog are three buttons: "OK", "Reset", and "Cancel".

8.4.3 Editing remote mirror locations

Each remote configuration must have at least one remote location specified. This is a list of remote hosts, and a list of the files that can be obtained. An existing remote location can be deleted or edited by selecting its name in the “Remote mirror locations” list in the “Remote configuration” editor. These can also be edited by adding, deleting or editing `RemoteLocation` objects in the `RemoteConfig.remoteLocations` attribute. Figure 8.11 shows the editor dialog for the “lion_srsdb” remote location:

Figure 8.11 Editing a remote location



The following properties can be edited (`RemoteLocation` attributes shown in parentheses):

Configuration name (*name*)

A descriptive single word used to refer to this location

Use as default location for mirroring (*default*)

If more than one location is specified for a mirror, this property indicates that if files have not been checked against the other specified locations, it should be checked against this. This allows a location to be specified as the “base-line” location for mirroring a directory.

Remote mirror hosts (*remoteHosts*)

A list of remote hosts which can be used as mirrors. These are specified in order of preference. If the first is not available, the second is used, and so on.

Files to mirror (*remoteFiles*)

A list of remote files which should be mirrored from this location.

8.4.4 Editing remote hosts

A remote host contains details of the location of a remote directory and how to connect to it. To add, delete or edit a remote host, select the entry in the **Remote mirror hosts** list of the remote locations editor, and click the appropriate button. Alternatively, `RemoteHost` objects can be added, edited or removed in the `RemoteLocation.remoteHosts` attribute. The same editor, shown in Figure 8.12 is used as for remote data and index files and is described in detail in Section 3.8.1.3, Specifying Remote Locations, page 56.

Figure 8.12 Editing a remote host

The screenshot shows the 'Remote Host Editor' dialog box. It has a title bar with a close button. The dialog is divided into three main sections: 'Remote location', 'SSL configuration', and 'Advanced Options'. In the 'Remote location' section, the 'Set individual server details' radio button is selected. Below it, there are fields for 'Protocol' (set to 'ftp'), 'Host' (set to 'downloads.lionbio.co.uk'), 'Port' (set to '21'), 'Username', 'Password', and 'Directory' (set to '/srs81/icarus/db/'). The 'SSL configuration' section has five rows, each with a label (SSL Certificate File, SSL Key File, SSL Configuration File, SSL Identity File, SSL Options) and a 'Browse' button. The 'Advanced Options' section has four rows: 'Failure retries' (set to 3), 'Retry sleep' (set to 30), 'Timeout' (set to 120), and 'Use proxy' (checked). There is also an 'FTP mode' dropdown set to 'binary' and a 'Content file' field with a 'Browse' button. At the bottom are 'OK', 'Reset', and 'Cancel' buttons.

Section	Field/Option	Value
Remote location	Set URL	<input type="radio"/>
	Set individual server details	<input checked="" type="radio"/>
	Protocol	ftp
	Host	downloads.lionbio.co.uk
	Port	21
	Directory	/srs81/icarus/db/
SSL configuration	SSL Certificate File	Browse
	SSL Key File	Browse
	SSL Configuration File	Browse
	SSL Identity File	Browse
	SSL Options	
Advanced Options	Failure retries	3
	Retry sleep	30
	Timeout	120
	Use proxy	<input checked="" type="checkbox"/>
	FTP mode	binary
	Content file	Browse

8.4.5 Editing remote files

A remote file contains details of the remote files of interest on the remote site and how they correspond to their local equivalents. To add, delete or edit a remote file, select the entry in the “Files to mirror list of the remote locations editor, and click the appropriate button. Alternatively, `RemoteFile` objects can be added, edited or removed in the `RemoteLocation.remoteFiles` attribute. The same editor, shown in Figure 8.13, is used as for remote data and index files and is described in detail in Section 3.8.1.3, Specifying Remote Locations, page 56, although not all possible attributes are shown.

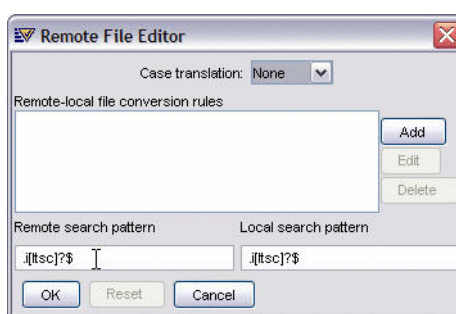


Figure 8.13 Editing a remote file pattern

Note: There is no support for unpacking compressed files - all configuration files are expected to be in their final form.

8.4.6 Editing memorized decisions

As described in Section 8.2.3, Conflict Resolution, page 225, SRS Prisma allows conflict resolution decisions to be memorized to allow automatic resolution of future conflicts involving specified objects, object attributes or files. These are added during the conflict resolution process, but can be edited at a later date. To add, remove or edit a memorized decision, select the appropriate entry from the **Automatic conflict decision** list in the remote configuration editor, and select **Edit**, **New** or **Delete** as

appropriate. Alternatively, the decisions can be edited directly as `ConflictResolution` objects in the `RemoteConfig.conflictResolutions` list. Figure shows the conflict resolution decision editor:



Figure 8.14 Editing a memorized conflict resolution decision

The following properties can be set:

Name

name of the file (e.g. `emblnew.i`), object (e.g. `EMBLNEW_DB`) or object attribute (e.g. `EMBLNEW.partSizeKb`). Note that when an object is selected, all conflicts involving that attribute will be automatically resolved.

Type

This can be set to file or Icarus object according to whether the entity involves a file or an object.

Action to take

This can be set to **Always preserve existing value** or **Always use new value** according to the decision to be made.

8.5 Trouble-shooting

If the process does not complete successfully, there are several different places in which information on the source of the problems can be found:

HTML reports

These are described in section 8.3 “Update Reports” (p. 229) and provide the most concise source of information on possible problems.

.log, .STDOUT and .ERROR files

These are the output produced by individual stages of the and may contain detailed errors that are not passed on to the main script, and hence can be very useful in checking problems.

The most common problem is lack of connectivity to the remote site. The settings for the download directories should be double-checked in `SRSLION/prisma`, and the error messages in `$SRSPRISMA/updateConfig/download.STDOUT/ERROR` should be checked for further clues to the problem. However, the same general settings are used as for the main SRS Prisma download mechanism, so there should be no need to reconfigure proxies etc.

Another problem that can occur is where the timestamp for local files has been changed, so new files are not downloaded. This should be resolved by resetting the timestamps on the local files to a sufficiently old date (e.g. using `touch -c -t 200001010101 $SRSD/*`). This may result in unnecessary updates, but should bring the installation up to date properly.

If this information does not help to resolve problems, please contact support@uk.lionbioscience.com, remember to keep a copy of the `$SRSPRISMA/configUpdate` directory so it can be used for debugging your problem

CHAPTER

9

SRS PRISMA - WORKED EXAMPLES

9.1 Introduction

SRS Prisma is a powerful and highly configurable system, which allows the Administrator to automate complex or tedious tasks. However, it is often easier to implement configurations by following worked examples. This Chapter provides a number of appropriate examples.

All the examples shown in this Chapter show how to directly modify the `Resource` object. The same implementation may be achieved by use of `VisAd`, as described in Chapter 3.

9.2 Single File Libraries

9.2.1 Simple Updating

In this section a simple library which uses a single file will be updated. The library being considered is an instance of `UNIPROT_SWISSPROT` where the flat-file is simply downloaded from a remote site with no need for any kind of unpacking. Example 9.1 shows parts of the `uniprot_swissprot.i` and `uniprot_swissprot.it` files respectively, used for a simple file download:

Example 9.1 Simple file download

Extract from uniprot_swissprot.i:

```
$UNIPROT_SWISSPROT_DB=$Library:[UNIPROT_SWISSPROT
  searchName:'uniprot_swissprot.dat'
  ...
]
```

Extract from uniprot_swissprot.it:

```
$UNIPROT_SWISSPROT_Res = $Resource:[
  updMethod:mirror
  remoteHosts:{
    $RemoteHost:[protocol:ftp
      host:"ftp.uniprot.org"
      downloadDir:"/pub/databases/uniprot/knowledgebase/"
    ]
  }
  remoteFiles:{
    $RemoteFilePattern:[
      searchPatternRemote:"uniprot_swissprot.dat"
      searchPatternLocal:"uniprot_swissprot.dat"
    ]
  }
]
```

The main point here is that a match between the files used for indexing (defined in `$Library.searchName`), the files downloaded from the remote site (`$RemoteHost.searchPatternRemote`) and the files used for comparison locally (`$RemoteHost.searchPatternLocal`) is ensured.

9.2.2 Unpacking Flat-Files

It is unusual to download simple, uncompressed files. In Example 9.2, the simple UNIPROT_SWISSPROT configuration is extended to support download and decompression of a gzipped file:

Example 9.2 Supporting download and decompression.

Extract from `uniprot_swissprot.it`:

```
$UNIPROT_SWISSPROT_Res = $Resource:[
  updMethod:mirror
  remoteHosts:{
    $RemoteHost:[protocol:ftp
      host:"ftp.uniprot.org"
      downLoadDir:"/pub/databases/uniprot/knowledgebase/"
    ]
  }
  remoteFiles:{
    $RemoteFilePattern:[
      searchPatternRemote:"uniprot_swissprot.dat.gz"
      searchPatternLocal:"uniprot_swissprot.dat"
      fromNamePattern:{'.gz'}
      toNamePattern:{''}
    ]
  }
  unpackCommand:{
    "$gunzip uniprot_swissprot.dat.gz"
  }
]
```

`searchPatternRemote` has been altered to reflect the name of the file wanted from the remote site, whilst `searchPatternLocal` remains the same. This allows the comparison of the unzipped local file with the gzipped remote file. In addition, to support the filename change that will occur, `fromNamePattern` and `toNamePattern` are used to notify SRS Prisma that the remote file name should have the string `.gz` removed.



Note: This type of renaming is normally carried out during download, but where the extension `.gz` or `.Z` is involved, the `.gz` or `.Z` extension is actually added back to ensure that any unzipping programs do not complain about incorrect extensions.

After downloading is complete, the `unpackCommand` is called. The filename is hardcoded, but any file pattern could be supplied here. The `gunzip` command is

provided as a variable, `$gunzip`, which is set in `$SRSSITE/srsdb.i` in order to allow the Administrator to set the local path to `gunzip`. This can be done with any variable desired in any string attribute, but where variable interpretation is used, a double quoted or bar string *must* be used.

The situation is usually a little more complex. The EBI FTP site provides the UNIPROT_SWISSPROT flat-file under the name, `new_seq.dat`, but this conflicts with the `$Library.searchName` definition. However, SRS Prisma supports file renaming, so the following `$Resource` class object directs renaming of the file `new_seq.dat.gz` to `uniprot_swissprot.dat`, as shown in Example 9.3:

Example 9.3 Extract from `uniprot-swissprot.it`

```
$UNIPROT_SWISSPROT_Res=$Resource: [name:UNIPROT_SWISSPROT
  updMethod:mirror
  remoteHosts:{
    $RemoteHost:[hostName:'ftp.uniprot.org' port:21
      downLoadDir:'/pub/databases/uniprot/knowledgebase/'
    ]
  }
  remoteFiles:{
    $RemoteFilePattern:[
      searchPatternRemote:'.*.dat.gz'
      fromNamePattern:'.gz' toNamePattern:''
    ]
  }
  unpackCommand:{
    "($gunzip) -f %s.gz"
  }
]
```

9.2.3 Reformatting Flat-Files

It is useful to be able to manipulate the data further, after download. In Example 9.4, a (hypothetical) command is added to `uniprot_swissprot.it` to create a new, reformatted file called `uniprot_swissprot.xtr` from the new `uniprot_swissprot.dat` file.

Example 9.4 Creating a reformatted file.

Extract from `uniprot_swissprot.it`

```
$UNIPROT_SWISSPOROT_Res=$Resource:[name:UNIPROT_SWISSPROT
  updMethod:mirror
  remoteHosts:{
    $RemoteHost:[hostName:'ftp.uniprot.org' port:21
      downloadDir:'/pub/databases/uniprot/knowledgebase'
    ]
  }
  remoteFiles:{
    $RemoteFilePattern:[
      searchPatternRemote:'*.dat.gz'
      fromNamePattern:'.gz' toNamePattern:''
    ]
  }
  unpackCommand:{
    "($gunzip) -f %s.gz"
  }
  reformatCommand:{
    "extractSeq %uniprot_swissprot.dat > uniprot_swissprot.xtr"
  }
  installFiles:{"uniprot_swissprot.xtr"}
]
```

Like `unpackCommand`, `reformatCommand` is a string containing the appropriate command. Note that `reformatCommand`, in this case, uses the `%d` placeholder to insert the location of the offline data directory for the `UNIPROT_SWISSPROT` library. This increases the portability of such definitions. Although the `reformatCommand` could be extended to move or further manipulate the swissnew FASTA file that has been produced, the `installFiles` command is used to add the `uniprot_swissprot.xtr` file to the list of files that should be moved from the offline to online data directories. `installFiles` can accept a list of filenames or file patterns, but these must always be present in the offline directory after downloading or rebuilding.

9.3 Handling Multiple Files

The examples earlier in this chapter have discussed updating of libraries using single files. However, most libraries use multiple files, and SRS Prisma is designed to apply parallelization where possible with such libraries. For example, UNIGENE currently uses multiple .data files. If we were to add a command to generate our hypothetical .xtr files from the UNIGENE .data files, the .i and .it files for this library might look like those shown in Example 9.5.

Example 9.5 Configuration for UNIGENE

Extract from unigene.i

```
$UNIGENE_DB=$Library:[name:UNIGENE
  parallelType:fileSize
  res:$UNIGENE_Res
  ..
]
```

Extract from unigene.it

```
$UNIGENE_Res = $Resource:[UNIGENE
  updMethod:mirror
  remoteHosts:{
    $RemoteHost:[hostName:'ftp.ncbi.nih.gov' port:21
      downloadDir:'/repository/UniGene'
    ]
  }
  remoteFiles:{
    $RemoteFilePattern:[
      searchPatternRemote:'.*.data.Z'
      searchPatternLocal:'.*.data' fromNamePattern:'.Z'
      toNamePattern:''
    ]
  }
  unpackCommand:"($gunzip) -f %s.Z"
  reformatCommand:"extractSeq %s > %duni_%n.xtr"
  installFiles:{"uni_*.xtr"}
]
```

The `.i` file shows that any file with the extension `.dat` can be used, and that files-based parallelization (`parallelType:filesSize`) should be used during indexing. With SRS Prisma, this also means that downloading and unpacking will also be parallelized. SRS Prisma is designed so that individual files are downloaded, unpacked and indexed independently. This means that the potential bottleneck of downloading will limit other processes as little as possible.

In the `.it` file the immediate difference to note is the use of the `%s` placeholder in `unpackCommand` and `reformatCommand` strings. Normally, these commands are only called once per database. However, when the `%s`, `%n` and `%e` placeholders are used, the command is executed once per file, with the full filename, basename or extension being added. The command is executed once per new file for `unpackCommand`, and once per file for `reformatCommand`. This means that the `gunzip` command will only be used on the files that have been downloaded during the SRS Prisma session, whilst the `extractSeq` command will be used on all the files used for indexing, including those that have not been updated. Also note the use of the construction `%duni_%n.xtr` to generate a new file name for the output file. This uses the offline directory, the string `uni_`, the basename for each file. So, this would take the filename `Zm.data` and generate a new filename, `uni_Zm.xtr`.

9.4 Dependent Libraries

9.4.1 A Simple Dependent Library

One of the most powerful aspects of SRS Prisma is its ability to add additional dependent libraries to carry out additional indexing or data manipulation. Example 9.6 is a simple dependent library (UNIGENEXTR) which indexes the XTR file generated by UNIGENE in the previous example.

Example 9.6 Indexing FASTA

Extract from `unigenextr.i`

```
$UNIGENEXTR_DB=$Library:[UNIGENEXTR
  format:$XTR_FORMAT
  res:$UNIGENEXTR_Res
  searchName:"uni_*.xtr"
]
```

Extract from srsdb.i

```
$LibLoc:[$UNIGENEXTR_DB
  dir:"$dataRoot/unigene/"
  offDir:"$dataRoot/unigene_tmp/"
  includeFiles:{"SRSSITE:unigenextr.i"  "SRSSITE:unigenextr.it"}
]
```

Extract from unigenextr.it

```
$UNIGENEXTR_Res=$Resource:[name:UNIGENEXTR
  updMethod:create
  dbDependOn:UNIGENE
]
```

UNIGENEXTR uses the same data directories as UNIGENE, and indexes the XTR files generated by the `reformatCommand` of UNIGENE, distinguishable by the `uni_` prefix. (The `$XTR_FORMAT` must be defined elsewhere.) It is declared as a dependent database by setting `updMethod` to `create` and `dbDependOn` to `UNIGENE` in the `$Resource` class object. The declaration of UNIGENEXTR as a dependent database means that UNIGENEXTR will be checked after UNIGENE has been checked and updated.

In this example, the data used by UNIGENEXTR is physically present after UNIGENE has been updated, but before UNIGENEXTR is checked. Hence, this data will be treated as new by SRS Prisma, and UNIGENEXTR will be indexed. In addition, as for any dependent database, the successful update of the parent library will force an update of UNIGENEXTR.

One last point should be considered here. Since SRS Prisma associates the XTR files created by the parent with UNIGENEXTR, they will be installed online by UNIGENEXTR. In this case, the `installFiles` attribute of UNIGENE should no longer include these files, as their presence would mean SRS Prisma would attempt to install them twice.

9.4.2 Dependent Libraries and File Information

Let us consider another example of the use of dependent libraries. Instead of the data used by a dependent library being created by the parents, imagine that a library exists that creates its own data during its update phase. Our next example considers

a library that executes a hypothetical `unpackCommand` to find the data files of a parent, and create GCG formatted copies which have the prefix 'emnewgcg_':

Example 9.7 Dependent Libraries

Extract from `srsdb.i`

```
$LibLoc:[$EMBLNEWGCG_DB
  enabled:y
  includeFiles:{
    "SRSSITE:emlnewgcg.it" "SRSSITE:emlnewgcg.i"
  }
  dir:"$dataRoot/emlnew/"
  offDir:"$dataRoot/emlnew_tmp"
]
```

Extract from `emlnewgcg.i`

```
$EMBLNEWGCG_DB=$Library:[EMBLNEWGCG
  format:$GCGSEQS_FORMAT
  res:$EMBLNEWGCG_Res
  searchName:"emnewgcg_*"
]
```

Extract from `emlnewgcg.it`

```
$EMBLNEWGCG_Res=$Resource:[
  updMethod:create
  dbDependOn:EMBLNEW
  unpackCommand:"find %EMBLNEW% -name \"*.dat\" -exec
    eml2gcg.sh {} \;"
  installFiles:{"emnewgcg_*"}
]
```

So, once EMBLNEW has been successfully updated, the update commands for EMBLNEWGCG are called, the new files are created, and indexed, then the data files installed online via `installFiles`.

However, the lack of file information for SRS Prisma about EMBLNEWGCG at the time of checking means that no parallelization, apart from type chunk, is possible, since no filename or size information is available for SRS Prisma to determine which `srsbuild` commands to issue when checking the library.

This situation can, however, be alleviated by the use of the `$Resource.useFiles` attribute. This adds information on file names (and sizes) to the library when the Administrator is confident that these will always be correct. When `useFiles` is set to `y`, the list of `LibFiles` in the `$Library.files` attribute is used by SRS Prisma to provide a dependent library with files-based parallelization, per-file unpacking or reformatting and flat-file installation. In this example, EMBLNEWGCG is assumed always to consist of the `emnewgcg_cum1`, `emnewgcg_cum2` and `emnewgcg_est1` files:

```
$EMBLNEWFASTA_DB=$Library: [EMBLNEWFASTA
  format:$GCGSEQS_FORMAT
  files:{
    $LibFile:emnewgcg_cum1
    $LibFile:emnewgcg_cum2
    $LibFile:emnewgcg_est1
  }
  res:$EMBLNEWGCG_Res
  parallelType:files
]
```

Extract from `emblnewgcg.it`

```
$EMBLNEWGCG_Res=$Resource:[
  updMethod:create
  dbDependOn:EMBLNEW
  useFiles:yes
]
```



Note: In this situation, the `installFiles` attribute should be removed from the EMBLNEW `$Resource` class object, since the dependent library installs the FASTA files generated by the `reformatCommand`.

It is also possible to use `installFiles` to provide file size information and hence enable parallelization of type `chunkSize` or `fileSize`, by setting the `size` attribute of each `$LibFile` class object. However, this should be used with caution because file sizes can change drastically between updates. However, the use of `usesFiles` imposes a degree of inflexibility and requires the Administrator to have advance knowledge of future situations. Fortunately, the `inheritFiles` attribute provides an alternative means of providing SRS Prisma with file information for a dependent library. For instance, our EMBLNEW database may be composed of a number of files

of unknown name and size, at the whim of the EMBL administrators. `useFiles` is not very useful here since we need to know filenames in advance. Instead, we can rely on the common processes applied to each file in the reformat stage of EMBLNEW.

For instance, it is possible to surmise, in this case, that the GCG filename is composed of the `emnewgcg_` prefix plus the `basename` of the parent, and that as a rule-of-thumb, the file size of the GCG file is roughly half that of the EMBL format file, e.g., EMBL format file `cum_1.dat` of size 100 Mb becomes GCG format `emnewgcg_cum_1` of size 50 Mb. This information can be used in the `$Resource` class object with the `inheritFiles` attribute to provide file information for SRS Prisma to use in writing indexing and data moving targets.

Example 9.8 Using a dependent library to index new data.

Extract from `embl.i`

```
$EMBLNEWGCG_DB=$Library:[EMBLNEWGCG
  format:$GCGSEQS_FORMAT
  searchName:"emnewgcg_"
  parallelType:fileSize
]
```

Extract from `emblnewgcg.it`

```
$EMBLNEWGCG_Res=$Resource:[
  updMethod:create
  dbDependOn:EMBLNEW
  inheritFiles:yes
  inheritFilePatternFrom:['[,$,.dat]']
  inheritFilePatternTo:['emnewgcg_,']
  inheritFileSize:0.5
]
```

It is hence possible to use `fileSize` parallelization because SRS Prisma is now provided with a reasonable estimate of the size of each generated file.

In conclusion, although `usesFiles` and `inheritFiles` can be readily used to provide SRS Prisma with additional file information, it is recommended that datafiles are created by parent libraries where possible. This minimizes potential for error where files with incorrect names or sizes are created.

9.4.3 Non-SRS Libraries for Data Manipulation

The combination of dependent libraries and inclusion of non-SRS libraries means that SRS Prisma can also be used to schedule data manipulation tasks linked to normal download and indexing tasks.

Example 9.9 illustrates the use of a non-SRS library and a dependent SRS library to process a tarball.

Occasionally, databases are distributed not as single flat-files, but as tar archives of larger files. This presents a challenge to the SRS Prisma philosophy of a correspondence between downloaded and indexed files. Fortunately, the use of dependent libraries can overcome this problem.

PATHWAY is a simple example, where all the component files are present in a directory structure within a large tarball. The current procedure for this database is to use a first, non-SRS library (PATHWAYTAR) to download and unpack the tarball, and a second dependent library (PATHWAY) to index the files produced by this unpacking. In this instance, the `unpackCommand` for the first database leaves the tarball untouched, so that it can readily be compared to the remote tarball, eliminating unnecessary downloads. Due its file-per-entry structure, the PATHWAY database has a single data directory, rather than separate online and offline directories.

Example 9.9 Processing a tarball, using a non-SRS library and a dependent SRS library.

Extract from `srsdb.i`

```
$LibLoc:[$PATHWAYTAR_DB dir:"$dataRoot/pathway/"  
  offDir:"$dataRoot/pathway/"]  
$LibLoc:[$PATHWAY_DB dir:"$dataRoot/pathway/"  
  offDir:"$dataRoot/pathway/"]
```

Extract from `pathway.i`

```

$PATHWAY_DB=$Library:[PATHWAY group:$PATHWAY_LIBS
  format:$PATHWAY_FORMAT
  maxNameLen:100
  searchName:'*'
]
$PATHWAYTAR_DB=$Library:[PATHWAYTAR group:$PATHWAY_LIBS
  format:$PATHWAY_FORMAT
  maxNameLen:100
  searchName:"*.tar.gz"
  type:hidden
]

```

Extract from pathway.it

```

$PATHWAY_Res=$Resource:[...
  updMethod:create
  unpackCommand:"echo 'indexing PATHWAY'"
  dbDependOn:'pathwaytar'
]
Extract from pathwaytar.it
$PATHWAYTAR_Res=$Resource:[name:PATHWAYTAR
  updMethod:mirror noSRS:yes
  remoteHosts:{
    $RemoteHost:[hostName:'ftp.genome.ad.jp' port:21
      downloadDir:'/pub/kegg/pathways'
    ]
  }
  remoteFiles:{
    $RemoteFilePattern:[
      searchPatternRemote:'pathway.weekly.last.tar.Z'
      searchPatternLocal:'pathway.weekly.last.tar.Z'
    ]
  }
  unpackCommand:{
    "cd %d; ($gunzip) -f -c %s | tar -xf -"
  }
]

```

Here, PATHWAYTAR is designated as a non-SRS database, which is never indexed. It is however of `type:hidden`, which means that it is not presented to users in the web interface.



Note: `$Library.searchName` and `$Library.format` are required attributes for the `$Library` class which are set to nominal values for `$PATHWAYTAR_DB` but do not affect handling by SRS Prisma.

`PATHWAYTAR` ensures that the `pathway.weekly.last.tar.Z` file is always up-to-date compared to the remote site. When a new version of this file is downloaded, the `unpackCommand` extracts the component files from the tarball whilst leaving it intact, saving them in the main pathway data directory. Generally this has the effect of adding new files to the `PATHWAY` data set.

9.5 Dependent Non-SRS Libraries

The previous examples illustrate further the concept of a non-SRS library being used to manipulate data before being indexed. Non-SRS libraries can also be used to format data further. An example might be to generate a non-redundant data set from multiple sequence databases, using a script that accepts a file containing a list of sequence files (Example 9.10).

Example 9.10 NRDB creation.

Extract from `nrdb.i`

```
$NRDB_DB=$Library:[
format:$NRDB_FORMAT
searchName:"nrdb*"
res:$NRDB_res
]
```

Extract from `nrdb.it`

```
$NRDB_Res=$Resource:[updMethod:create
dbDependOn:{'UNIPROT_SWISSPROT' 'UNIPROT_TREMBL'}
noSRS:yes
unpackCommand:|cd %d; rm -f inputList; \
                |find %UNIPROT_SWISSPROT% > inputList; \
                |find %UNIPROT-TREMBL% >> inputList; \
                |createNrdb -fileList inputList > nrdb\
```

```
    reformatCommand:"cd %d; formatdb -i nrdb -t nrdb -p T"  
    installFiles:{"nrdb" "nrdb.p*"}  
  ]
```

This dependent library illustrates the use of multiple parents - this library will be updated, and its commands run whenever one of the named parents is updated. However, the use of the %DBNAME% placeholders in the `unpackCommand` allows the file list needed for NRDB creation to be assembled from the online and offline files as appropriate, according to which files have been updated. The files created by the `unpackCommand` and `reformatCommand` are then installed online as specified by `installFiles`.

CHAPTER

10

SRS PRISMA - COMMON ERRORS

10.1 Introduction

The advice in Chapter 6, SRS Prisma - Troubleshooting should be sufficient to aid rapid error diagnosis, but there are a number of errors that can be made during configuration that are easy to spot. This chapter describes how to tackle some of the more common problems that may be encountered.

10.2 Problem: Changes to a Resource Object Are Not Recognized

The customized `.it` file that contains the `Resource` object must be referenced in the `LibLoc` object for the library, and the appropriate resource object should be named in the `Library` object.

For example:

```
from srsdb.i

$LibLoc:[$EMBLNEW_DB includeFiles:{"SRSSITE:emblnew.it" "SRSDB:embl.i"} ...]
from embl.i:
$Library: [EMBLNEW
..
res:$EMBLNEW_Res
..
]

from SRSSITE:emblnew.it
$EMBLNEW_Res: [
..
]
```



Note: Changes to individual attributes of Icarus objects (e.g. in `site.i`) are no longer supported. Instead, the file containing the modified `Resource` object should be copied to `$SRSSITE` and `srsdb.i` updated to reflect this change.

10.3 Problem: SRS Prisma Tries to Move a Non-Existent File Online and Fails

This is commonly because one of `installFiles` or `usesFiles` has been used to specify one or more files that have not been downloaded or created during each update. The names of files that have been created should be checked to ensure they match those used, and the files named should always be created/downloaded. If files are not always used, note that the `reformatCommand` can be used to run external commands that move files online conditionally.

10.4 Problem: The Downloading Fails Frequently because the FTP Connection is Unreliable

The easiest solution is to switch to a more reliable or more local mirror site, but if this is not possible, the number of retry attempts can be increased to allow for interruptions, the time-out interval can be increased, and the time to sleep between retries can also be increased. Please consult 3.8.1.9 “General Options”, p. 66 for more details on changing these values for a specified remote host.

10.5 Problem: SRS Prisma Reports that Files Need to be Updated, but Downloading always Fails

This is commonly due to incorrect write permissions on the offline data directory.

10.6 Problem: The Unpack Command Contains a Mangled Filename, e.g. `gunzip uniprot_sprot.dat.gz.gz`

This is usually due to the absence of conversion rules between the remote and local files. In this case, a file conversion pattern mapping “.gz” to “” should be added to inform SRS Prisma that the .gz extension is removed during the download and unpack process. Please consult Section 3.8.1.10, Specifying Remote Files, page 67 for more details on configuring this option.

10.7 Problem: SRS Prisma Tries to Run srsbuild Commands with `–parts 0` on a Dependent Library

This is usually due to use of files-based parallelization where there is no file information. `usesFiles` or `inheritFiles` should be used here (see Section 3.14.8, Dependent Library File Lists, page 124), or no parallelization should be specified.

10.8 Problem: SRS Prisma Keeps Trying to Rebuild the Library Even Though No New Files Are Downloaded

This is often because there are non-indexed files in one of the data directories that are not found on the remote site and hence must be removed, e.g. FASTA files, BLAST indices. Check the Decision report or the file `$SRSPRISMA/<run>/<DBNAME>/<DBNAME>.removed`, and alter `searchName` or `localSearchName` in the `$Resource` class object to limit the files compared.

10.9 Problem: All the Files for the Library Have Been Deleted by SRS Prisma

This is frequently because the files being compared remotely and locally do not match. Check the regular expressions provided in the remote file configuration (see Section 3.8.1.10, Specifying Remote Files, page 67). Also check that the `RemoteHost.downloadDir` is correct. `$SRSPRISMA/<run>/<DBNAME>/<DBNAME>.remote` will indicate which files are present on the remote site.

10.10 Problem: SRS Prisma Reports that No Suitable Files for Indexing Are Found

An error of the following form can sometimes be seen in the output of the check phase for an SRS Prisma run:

```
"[Warning] no suitable files for SRS indexing found for library "ENZYME"
```

(This may reflect a discrepancy between the `searchNames` specified for use by PRISMA in the `$Resource` object and the `filenames` specified for use by SRS indexing in the `$Library` object)

This error is usually caused by a mismatch between the files being downloaded (e.g. those found on the remote site that match the regular expression specified in the `RemoteFilePattern` object) and the files required for indexing by the `Library` object (e.g. `Library.searchName`). This can cause serious problems with the update and installation if a files-based parallelization method is used e.g. `filesSize`.

To fix this problem, ensure that the `Resource` object is configured to download files that, after any download renaming, match the `searchName` or files specification for the library.

If this problem occurs with a dependent library, it may be that the required files are generated during execution of pre-processing commands. This error can be ignored if the `Resource` object specifies `separate` and if non-files parallelization is used. If this is not the case, `Resource.useFiles` or `Resource.inheritsFiles` should be used to specify the files involved (see Section 3.14.8, *Dependent Library File Lists*, page 124).

10.11 Problem: Remote Checking Fails for Some Libraries

The following type of warning may be seen in the check phase output:

```
Cannot check remote files for library "LIBNAME"  
Checking database files locally...
```

There are a number of possible causes, but one of the most common is where the e-mail address is not set correctly for the installation. This address is used as the password for anonymous FTP logins, and is set to 'unknown' by default. However, many public FTP servers (e.g. <ftp.ebi.ac.uk>) do not accept this as a valid e-mail address. To change this, simply edit the following line in `$SRSICA/wwwlocal.i` and run `srssection`:

```
$adminEmailPar = $Parameter:[str val:"mailto:unknown" isVolatile:1]
```

to

```
$adminEmailPar = $Parameter:[str val:"mailto:<valid email>" isVolatile:1]
```

If this does not solve the problem, it is advisable to check the contents of following files from `$SRSPRISMA/<run>/flags/<DBNAME>`:

- `<DBNAME>.check.STDOUT` (output from remote check)
- `<DBNAME>.check.ERROR` (errors from remote check)

APPENDIX

A

WGETPRISMA

A.1 GNU wget

GNU `wget` is a free software package for retrieving files using HTTP, HTTPS and FTP. The homepage for GNU `wget` is <http://www.gnu.org/software/wget/wget.html>.

SRS Prisma uses a modified version of `wget` to provide support for checking and downloading files from remote FTP, HTTP and HTTPS sites. Section A.2, Modifications to `wgetPrisma`, page 268 explains what modifications have been made. The modified source code is supplied with SRS under the `$SRROOT/gnusrc/wget` directory.

A.2 Modifications to `wgetPrisma`

To support the features needed by SRS Prisma, the following modifications have been made to `wget` 1.9.1:

Large file support

`wget` does not support downloading of files larger than 2Gb on all platforms. `wgetPrisma` has been modified to support use of these files.

Extended FTP proxy support

The support provided by `wget` for FTP proxies is very limited, and `wgetPrisma` has been extended to support 10 different types of FTP proxies. The following additional command line flags are provided:

- `--ftp-proxy-type` - type of proxy to use (supply number from 1-10)
- `--ftp-proxy` - use specified URL for FTP proxy
- `--http-proxy` - use specified URL for HTTP proxy

Report-only functionality

`wgetPrisma` has been extended to provide plain text listing of files from specified FTP and HTTP locations with minimal downloads. This is activated by using the `--report-only` flag, which takes as an argument the name of the file to write the report to.



Note: The shipped version of `wgetPrisma` is compiled against OpenSSL v0.9.7d to support HTTPS. `wgetPrisma` can be recompiled against an alternative SSL library, or compiled without SSL support. The next section explains how.

A.3 Compiling `wgetPrisma`

Pre-compiled binaries of `wgetPrisma` are supplied with SRS, but source code is also supplied so that it can be recompiled locally if required. To recompile `wgetPrisma`, follow these steps:

1. Enter the `$SRSROOT/gnusrc/wget` directory
% `cd $SRSROOT/gnusrc/wget`
2. (optional) Remove previous configuration files
% `make distclean`
3. Configure the compilation process:
% `./configure`

`./configure` takes a variety of arguments, including:

- `--without-ssl` to compile without SSL support
 - `--with-ssl=PATH` to compile against SSL libraries located under `PATH`
4. Compile and install `wgetPrisma` (default installation location is `$SRSEXEC`)
% `make`
% `make install`

A.4 Use `.wgetrc`

`wget` allows general settings to be stored in a configuration file. The configuration file is defined with the environment variable `WGETRC`, or alternatively `$HOME/.wgetrc` is used. `wgetPrisma` will also read this file, but since SRS Prisma automatically supplies certain values on the command line, not all settings will be honoured, and some settings are not compatible with `wgetPrisma`. The following table shows which options are permitted, and under what circumstances. All others should not be

used as they either conflict with SRS Prisma's use of `wgetPrisma`, or are not appropriate.

Table 0.1 Permissible wgetPrisma configuration

Command	Permissible?
ftp_proxy	Yes, if proxy settings not already defined
http_proxy	Yes, if proxy settings not already defined
http_proxy_user	Yes, if proxy settings not already defined
http_proxy_password	Yes, if proxy settings not already defined
proxy_user	Yes, if proxy settings not already defined
proxy_password	Yes, if proxy settings not already defined
use_proxy	Yes, if proxy settings not already defined
sslcertfile	Yes, when not defined for a library
sslcertkey	Yes, when not defined for a library
egd_file	Yes
sslcadir	Yes
sslcafile	Yes
sslcerttype	Yes
sslcheckcert	Yes
sslprotocol	Yes
passive_ftp	Where passive FTP not set for a library
user_agent	Yes
random_wait	Yes
load_cookies	Yes
connect_timeout	Yes
read_timeout	Yes

Command	Permissible?
wait	Yes
bind_address	Yes
limit_rate	Yes
dns_cache	Yes
randomwait	Yes
use_cookies	Yes

To use an alternative configuration file, the `WGETRC` environment variable should be set to the location of the file. This can include an empty file if no configuration is required but `$HOME/.wgetrc` contains conflicting configuration

INDEX

A

archiving	
SRS Quality Report	212
attribute	
checkFileSizes	78
completedFile	44

B

bsub flags	36
------------	----

C

checkFileSizes attribute	78
checkLocalOnly parameter	43
Completed flag	44
completedFile attribute	44

D

databank group reports, SRS Quality Report	204
--	-----

E

error details page, SRS Quality Report	207
error details, SRS Quality Report	209
error summary report, SRS Quality Report	208

F

file name substitution	73
------------------------	----

M

monthly errors breakdown, SRS Quality Report	210
--	-----

Q

quick references, SRS Quality Report	209
--------------------------------------	-----

S

SRS Quality Report	
archiving	212
description of	194
HTML pages	
databank group report	204
error details page	207

error details, full	209
error summary report	208
monthly errors breakdown	210
quick references toolbox	209
summary report	202
tests used	212
summary report, SRS Quality Report	202

T

tests, SRS Quality Report	212
---------------------------	-----